ACTS: Attestations of Contents in TLS Sessions

Pierpaolo Della Monica, Ivan Visconti, Andrea Vitaletti and Marco Zecchini
Sapienza University of Rome, Italy
{dellamonica, visconti, vitaletti, zecchini}@diag.uniroma1.it

***Abstract*—An essential requirement for the large-scale adoption of Web3 is enabling users to benefit from their data even within already deployed systems. This raises an important open question: how can existing, widely adopted software verify that a user has retrieved specific data from a TLS server?**

Impressive scientific results (e.g., DECO [CCS20] and the work of Xie et al. [USENIX24]) and industrial products (TLSNotary) have recently made progress in the above challenging direction. However, while they nicely leave TLS servers untouched, the retrieved data is then used in computations with verifiers that are required to run some advanced non-standardized cryptographic schemes (e.g., ZK-SNARKs), which clearly limit the large-scale adoption of the proposed technologies.

In this paper, building on top of previous approaches and relying on the recent concept of Predicate Blind Signatures of Fuchsbauer and Wolf [Eurocrypt24], we bypass the limits of prior work by presenting ACTS, a distributed architecture that, while still leaving TLS servers untouched, it allows a user to show possession of data retrieved from TLS servers simply requiring that the software of the verifier can check a standard signature.

Our contributions include a round-optimal predicate blind signature protocol that produces standard RSA-PSS signatures. We show how this primitive can be integrated into the DECO architecture (and its successors) to certify data retrieved from TLS servers. Furthermore, we have optimized our construction to make it practical on commodity hardware for a large and significant class of policies implemented by the notary (i.e., the actor that is in charge of obliviously certifying TLS data, therefore preserving data confidentiality).

We provide an experimental evaluation on the simple but powerful enough use case of a PDF document downloaded from a TLS server and encoded into an AES-GCM ciphertext. The user will then get a certified PDF through a standard PADES signature added obliviously to the PDF along with some metadata by a notary service. The resulting standard signed PDF document can be transparently verified using off-the-shelf PDF readers. Our experimental validation demonstrates that our architecture is suitable for real-world deployment in concrete scenarios.

I. Introduction

Nowadays, user data are stored in many servers on the Web. Typical examples include personal health and banking data, as well as financial or registry data. Users can usually read and download their data after an identification phase, and the entire communication is typically secured by the Transport Security Layers (TLS) protocol so that the communication is both confidential and authenticated.

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.241861
www.ndss-symposium.org

However, it is very common that data downloaded from servers is not certified. For example, one can download from the server of a medical laboratory a PDF, with the results of a blood test, and this file is commonly not digitally signed. This example can be easily replicated to many other scenarios, and the TLS protocol — by design — does not leave the client any useful information to convince others of the authenticity of the data that has been downloaded from the server.

The consequence of the above state of affairs is that users cannot provide guarantees on the authenticity of their own data unless they embark in time and money consuming requests of ad-hoc certifications, which moreover, are not guaranteed to succeed since servers might not be willing to cooperate.

The birth of TLS Oracles. Starting in 2020, the seminal paper called DECO [1], initializes a major line of research [2, 3, 4][1] in which a third party, named verifier or TLS oracle, audits the communication between the user and the server. In particular, in a first phase, the verifier participates, jointly with the user, in a TLS handshake phase where they jointly play the role of a client and connect to a TLS server obtaining shares of the session keys that will be used to secure the communication of application data. Next, in a second phase, using the above shares, they jointly produce packets of the TLS record layer (e.g., encrypted and authenticated HTTP packets) that are sent to the server. In this second phase, the verifier does not learn anything about the content of the HTTP packets sent to the server, thus maintaining the confidentiality of user data. Once the communication with the server has ended, the user has the ability to send a cryptographic proof (specifically, a zero-knowledge proof) demonstrating that the exchanged application-layer data (e.g., HTTP packets) satisfy certain properties (i.e., enabling the user to prove to a verifier properties about the content of a TLS connection established with a server). For instance, if the server is the website of a bank, the user can prove to the verifier that she has an account with this bank and that, in this account, she holds more than a specific amount of money.

A fundamental key feature of the protocol proposed in [1], called DECO, lies in the fact that it requires no changes on the server side, which continues to work exactly as it does today (i.e., there is no need to modify the widely deployed TLS protocol and the entire process is transparent to the server). The above transparency is extremely beneficial and indeed some companies, such as the decentralized oracle Chainlink (https://blog.chain.link/deco-sandbox/), are already

[1]TLSNotary and PrimusLabs are industrial projects working in this field.

using DECO (or variants of it), thanks to its easy integration with existing services.

A weakness of DECO. The work of Zhang et al. [1] (and subsequent optimizations [2, 3]) considers three parties: a server, a user and a verifier. The user should repeat executions of DECO with every distinct verifier and for every distinct session engaged with the server. Considering the fact that DECO introduces a significant overhead, the above workflow is clearly impractical for large-scale adoption of scenarios where the user needs to engage with multiple verifiers.

TLSNotary (https://tlsnotary.org), an open-source project that works in this field, proposes a technique to mitigate such a problem by introducing an additional player, called a *notary*. The notary replaces the verifier in DECO and, once the communication with the server concludes, she signs the packet of the record layer of TLS, corresponding therefore to authenticated encryptions of application data, which typically consist of HTTP packets exchanged at the application layer between client and server.

Those ciphertexts signed by the notary can include confidential information of the user and obviously the ciphertexts protect such confidentiality since the notary does not possess the decryption key. A verifier who trusts the notary first verifies that the signature on the encrypted HTTP packets is correct and, then, receiving the decrypted application data and the decryption key from the user verifies that the encrypted HTTP packets are correctly decrypted into the application data. Note that, with this approach, the notary can collude with a verifier understanding in which session she has signed the HTTP packets and giving her the chance to leak other information about the user. If the notary repeats this process with many users, this would transform her into a sort of big brother.

To avoid this attack, with TLSNotary, the user must provide a zero-knowledge proof to the verifier, as in DECO, guaranteeing that the data in the hidden plaintext satisfy some properties (for a visual intuition of this process, we refer the reader to Fig. 1 and to Sec. I-C). However, such a verification is typically way beyond what is available in existing systems, where software is limited to standardized cryptographic features, such as data encryption and digital signatures. For instance, Adobe Acrobat Reader is one of the most widely adopted software used to read PDF files, and it does not include any implementation of verification algorithms of zero-knowledge proofs, while it does support the verification of standard digital signature schemes attached to PDFs. The need to upgrade the verifier with non-standardized cryptographic tasks is a serious limitation to the applicability of such techniques in real-world scenarios and therefore negatively affects the adoption of DECO and all subsequent protocols [2, 3, 4].

Recall that a key feature of DECO is the transparency for servers that are not required to update software and to support non-standardized cryptographic tasks. Clearly, one would also like the same transparency for verifiers, but proposals in the state-of-the art are instead "unfriendly" to verifiers.

Notice that one cannot just transfer from the verifier to the notary the task of verifying a ZK proof since the claim of the proof can already include confidential information that the user is not willing to reveal to the notary. Indeed, providing to the notary all claims that users would like to prove to verifiers would again transform the notary into a sort of big brother collecting data from all users, which is obviously undesirable.

Open Problem. In the current state of affairs, there is a clear tension between allowing servers and verifiers to keep working using existing software consisting of *standardized cryptographic tools* (e.g., digital signatures) only, and the need to allow a user to show possession of data stored on servers through a notary, still preserving data *confidentiality* with respect to the notary. The open problem that we study in our work consists of designing a secure protocol that circumvents the above tension while guaranteeing the following properties:

- a user can convince verifiers about possession of data stored on servers leveraging a notary service;
- there is no leak of information about user data that can be abused by the notary;
- the system is completely transparent to servers;
- verifiers only need to use existing software based on standardized cryptographic tools.

A. *Our results*

In this work, we solve the above open problem. We present **ACTS** (Attestations of Contents in TLS Sessions), a new architecture that builds upon prior approaches leveraging notaries, but we deviate in some crucial steps allowing verifiers to rely solely on existing software that can verify standard signatures.

At the heart of our approach there is the use of an advanced cryptographic building block that, however, will impact only on user and notary: predicate blind signatures for standardized signature schemes.

Brief warm up on blind signatures. Introduced by Chaum in [5], a blind signature scheme is a protocol between a signer and a user that allows the latter to obtain a signature on a message hidden from the signer. Due to their applications (e.g., e-cash systems [5], e-voting [6], anonymous credentials [7], blockchain applications [8]), recently the IETF has realized a proposal [9] for the standardization of blind signatures over the standard signature scheme RSA-PSS [10, 11].

Blind signatures are not enough. In our application, we propose to embed an execution of a blind signature protocol in the communication between the user and the notary so that the user gets from the notary (playing as signer) a standard RSA-PSS signature on the actual application data (e.g., HTTP packets) without the notary having access to such data. The verifier can then simply check the signature of the notary on the downloaded data with standard software. For instance, if the user, collaborating with the notary, downloads a PDF file from a server, the notary can blindly sign such a file and the verifier can check the signature with standard software, like Adobe Acrobat Reader. However, this approach raises a natural concern: the notary does not trust the user and obviously she would refuse to sign arbitrary hidden data.

Upgrading to predicate blind signatures. To overcome the above barrier, we use the recent notion of *Predicate Blind Signature* (PBS) proposed by Fuchsbauer and Wolf [12]. A PBS scheme generalizes the notion of a blind signature, allowing the signer to produce a signature on a hidden message only if the message satisfies some specific properties (i.e., it satisfies a predicate) established by both the user and the signer. This notion perfectly fixes the above problem where a notary was not willing to sign arbitrary data. Indeed, we can use a PBS scheme so that the notary is guaranteed that the signature will be applied exactly to some application data included in the ciphertext (e.g., a PDF downloaded by the user) without being able to see the signed data.

Since our goal is to allow the verifier to use existing software, including standardized cryptographic tools only, we have designed and implemented a predicate blind signature scheme that outputs signatures according to the standard and widely adopted RSA-PSS scheme. We have used our implementation to blindly sign a PDF encrypted in an AES-GCM ciphertext, producing a signed PDF according to the standard PADES. The widely used software Adobe Acrobat Reader successfully shows the content of the PDF file and moreover confirms that the PDF is correctly signed by the notary.

Another layer of security: decentralization of the notary. One might worry about specific scenarios where collusion between the user and the notary is possible (e.g., when the content of a TLS communication has a relevant value and the user can therefore try to bribe the notary).

We notice that we can protect the verifier by strongly mitigating the above risks through the following requirement: the user must perform the task of obtaining signed data from a notary multiple times, each time accessing a different notary. We envision a setting in which, to avoid a single point of failure, the verifier requires the user to provide data signed by multiple notaries. The user, therefore, will produce in the end a predicate blind multisignature. Notice that the notion of blind multisignature has recently been introduced in [13].

A natural question is why we choose to rely on blind multisignatures rather than threshold blind signatures, especially given that the latter can yield a single signature. To answer this, we first need to discuss the key differences between these two primitives. In both notions, the setting is defined for a single user and multiple signers. In a threshold blind signature scheme, signers collaboratively generate a single blind signature, with the requirement that at least t out of n parties participate (with $t \leq n$). In contrast, in a blind multisignature scheme, each signer independently interacts with the user to produce a blind signature for the same message, and then these signatures are aggregated by the user.

Although constructions of threshold blind signature schemes exist [14, 15], it is well known that these schemes typically lack *adaptive security*. Adaptive security in this context refers to the property of the scheme to remain secure (i.e., unforgeability holds) even in the case that the malicious user can corrupt some signers during the execution of the protocol. Note that in a real-world context, as the one we proposed before, such a scenario where a user can adaptively corrupt some notaries could be realistic, especially in the case where the content of the TLS communication to certify has a relevant value. That is, for our purpose, we cannot give up on adaptive security, even though there is currently no scheme that supports such an advanced security notion. Furthermore, even if one attempts to construct (for the first time) an adaptively secure threshold blind signature scheme, in our case, there is also another additional challenge that the output must be a standard (e.g., RSA-PSS) signature; this is another major challenge to overcome since standard signatures are not easily thresholdizable (e.g., like BLS signatures).

Given these challenges, we opt for (predicate) blind multisignatures, providing stronger security at the affordable cost of attaching multiple signatures to the final output. This choice, however, comes with a notable advantage: it enables modularity. Specifically, different verifiers may trust different subsets of notaries, and our approach naturally supports this by allowing each verifier to extract and validate only the signatures corresponding to the notaries it trusts.

Moreover, based on our experimental results, we observe that the size overhead required for including multiple signatures is minimal relative to the size of the underlying content (e.g., the content of an HTTP communication). In practice, this keeps our approach efficient and practical, even when a reasonable number of notaries are involved.

B. Related Works on TLS Oracles

DECO [1], initially derived from the PageSigner protocol (2014), formalizes the problem of proving the provenance of TLS data without requiring any server-side modifications. It introduces the first protocol in which the user and the verifier jointly emulate a TLS client through an ad hoc two-party computation (2PC) in the malicious setting. At the end of the protocol, the user uses a Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (zkSNARK) to prove statements about the data received from the server.

The work of Xie et al. [2] improves the efficiency of DECO. Their key idea is that they do not change the overall architecture of the DECO protocol but instead optimize its internal components. In particular, instead of designing custom maliciously secure 2PC protocols as done in DECO, the prover (i.e., the user) sends a garbled circuit to the verifier, who evaluates it. After the user and verifier interact jointly with the server and when the communication phase ends, the verifier reveals its share of the session secrets. At this point, the prover proves in zero-knowledge (using a highly efficient interactive protocol based on VOLE [16]) that the garbling was carried out honestly and consistently. This approach is named *garble-then-prove* and significantly reduces both the computation and communication overhead of DECO. DiStefano [17] and DIDO [18] further optimize the security and efficiency of DECO [1] especially for TLS 1.3 version.

Janus [4] builds on the alternative setting described in [1, App. C.4], where the verifier acts as a *proxy* between the user and the server. In this model, the 2PC is performed only during the TLS handshake. After this phase, the verifier reveals its key share to the user, who then constructs HTTP requests independently. These requests are forwarded by the verifier to the server, effectively reducing the number of 2PC executions. This proxy-based approach significantly relies on stronger network assumptions: in particular, the user cannot perform a man-in-the-middle attack on a network level between the verifier and the server. Janus integrates the garble-then-prove paradigm into the handshake phase of the TLS 1.3 version, enhancing the efficiency of the protocol.

Always in the proxy setting, Origo [3] eliminates the need for 2PC entirely, even during the handshake. It exploits specific properties of TLS 1.3 to allow the user to generate a SNARK after the handshake phase with the server, proving that she acts honestly during this phase. This results in constant-size communication during the online phase and makes the protocol highly suitable for constrained environments or scenarios involving multiple verifiers. However, Origo still relies on the above network assumption, limiting the adversarial user from being a proxy between verifier and server. Luo et al. [19] designs a protocol with the same objective of Origo but without relying on SNARK but instead it directly explots the key-commiting property of the authenticated encryption with associated data schemes used in TLS. However, according to [4], such an approach might have limitations in terms of security. We refer the reader to [4, Table 5] and [3, Table 1] for a comparative summary of the results and key characteristics achieved by these systems.

Although the proxy-based setting adopted by Janus and Origo leads to significant efficiency improvements, we do not follow this approach in our work. As early noted in [1], relying on the network assumption that the user cannot interfere with the communication between the verifier and the server weakens the security model. For this reason, we focus exclusively on designs that do not depend on a proxy and, therefore, on the above network assumption.

DECO [1] and the works [2, 3] considers three parties: a user, a verifier, and a server. TLSNotary (https://tlsnotary.org) is the first project to identify the following problem: the user should repeat the executions of DECO (or alternative protocols) with every distinct verifier and for every distinct session engaged with the server. To fix this problem, TLSNotary introduces a fourth party, a notary, that acts as the verifier in DECO and, at the end of the protocol, signs the ciphertexts of the HTTP packets. A verifier who trusts the notary first verifies that the signature on the encrypted HTTP packets is correct and, then, verifies a zero-knowledge proof received from the user, as in DECO, that guarantees that data in the hidden plaintext satisfy some properties. Janus [4] also considers a four-party setting with a notary. TLSNotary is an open source project that implements all the technical components of the protocols using a 2PC in the malicious setting (specifically, the Dual Execution with Asymmetric Privacy protocol) for the handshake and the query phase and a NIZK to prove properties on the hidden plaintext.

C. Technical Overview

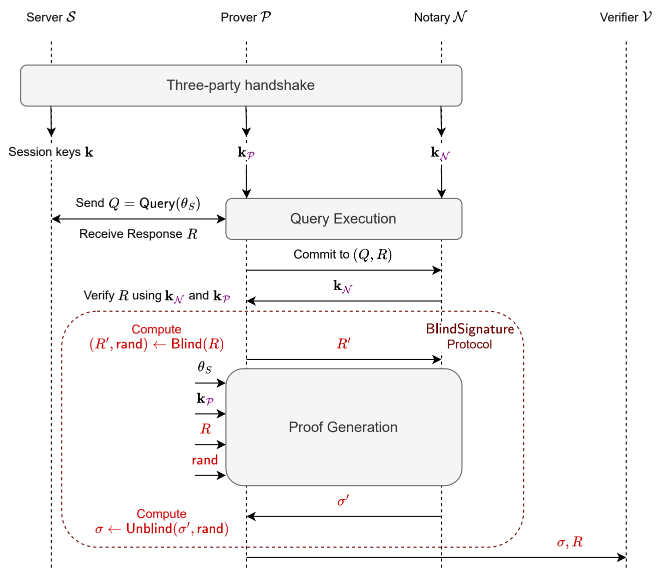

Server $\mathcal{S}$
Prover $\mathcal{P}$
Notary $\mathcal{N}$
Verifier $\mathcal{V}$
Three-party handshake
Session keys $\mathbf{k}$
$\mathbf{k}_{\mathcal{P}}$
$\mathbf{k}_{\mathcal{N}}$
Send $Q = \mathsf{Query}(\theta_S)$
Receive Response R
Query Execution
Commit to (Q, R)
$\mathbf{k}_{\mathcal{N}}$
Verify R using $\mathbf{k}_{\mathcal{N}}$ and $\mathbf{k}_{\mathcal{P}}$
Compute
$(R', \mathsf{rand}) \leftarrow \mathsf{Blind}(R)$
R'
BlindSignature Protocol
θ_S
$\mathbf{k}_{\mathcal{P}}$
R
rand
Proof Generation
σ'
Compute
$\sigma \leftarrow \mathsf{Unblind}(\sigma', \mathsf{rand})$
σ, R

Fig. 1: An overview of **ACTS**. Note that, we highlight in red the deviation of our solution with respect to [1]. The last step can also be interactive and played subsequently.

Starting point: DECO. We position our work as a challenging extension of DECO as in [1] that aims at obtaining a notary service[2] from DECO with the purpose of allowing a user to prove to verifiers[3] that she own data stored by TLS servers and certified by a notary. Such verifiers use existing software with standardized cryptography only.

Therefore, in order to illustrate our extension, we first need to show how DECO works (having a notary replacing the DECO verifier) and then we will present an overview of our extension. Note that other related works on TLS oracles (presented in App. I-B) follow a similar architecture presented in the following with only small modifications.

As shown in Fig. 1, which is inspired by Fig. 2 of [1], DECO is a three-phase protocol. The first phase is a three-party handshake protocol in which the prover (or user) $\mathcal{P}$, a notary $\mathcal{N}$, and the TLS server $\mathcal{S}$ establish session keys that are secret-shared between $\mathcal{P}$ and $\mathcal{N}$. At the end of the three-party handshake, $\mathcal{P}$ and $\mathcal{N}$ receive $\mathbf{k}_{\mathcal{P}}$ and $\mathbf{k}_{\mathcal{N}}$, respectively, while $\mathcal{S}$ receives $\mathbf{k} = \mathbf{k}_{\mathcal{P}} + \mathbf{k}_{\mathcal{N}}$.

After the handshake, there is a query execution phase during which $\mathcal{P}$ accesses the server following the standard TLS protocol, but with the help of $\mathcal{N}$. In the query execution phase, $\mathcal{P}$ and $\mathcal{N}$ agree on a query template $\mathsf{Query}(\cdot)$ that will be sent to $\mathcal{S}$. In the query template, the parties agree on the public parameters of the web request to $\mathcal{S}$ (e.g., a URL for a REST API server), but additionally, it can be fulfilled with a secret string θ_S known by $\mathcal{P}$. Hence, $\mathsf{Query}(\cdot)$ takes as input θ_S. An example query template would be the URL

[2]The notary will play on the side of the verifier of DECO.

[3]They are verifiers in the 4-party model, they interact only with the user and this interaction can happen when the execution of DECO is over already.

of a REST API that returns the stock price of TSLA on a specific day, and, in this case, θ_S would be the API key to access such information on $\mathcal{S}$ (e.g., $\mathsf{Query}(\theta_S)$ = "https://eodhd.com/api/eod/TSLA.US?from=2025-01-01&to=2025-01-02&period=d&fmt=json&api_token=θ_S"). Note that, for some applications, it is relevant only that $\mathcal{P}$ has been able to download application data R from $\mathcal{S}$ without checking whether the query to $\mathcal{S}$ was well-formed. During the query execution phase, $\mathcal{P}$ and $\mathcal{N}$ jointly compute the authenticated encryption of the query $Q = \mathsf{Query}(\theta_\mathsf{S})$ that is then sent to $\mathcal{S}$ by $\mathcal{P}$. The server $\mathcal{S}$ will respond with the authenticated encryption of R to $\mathcal{P}$. When the communication with the server ends, $\mathcal{P}$ commits to the query and response (i.e., $\mathcal{P}$ shares with $\mathcal{N}$ the ciphertext of R and Q) and $\mathcal{N}$ reveals her key share. Finally, $\mathcal{P}$ proves statements about the response R in a proof generation phase. During this phase in DECO, the prover proves in zero-knowledge to the notary that (1) Q was well-formed (i.e. $Q = \mathsf{Query}(\theta_\mathsf{S})$), (2) $\mathbf{k}_\mathcal{P}$ is combined with $\mathbf{k}_\mathcal{N}$ into a key $\mathbf{k}'$ that correctly decrypts the authenticated encryption of R and Q, and finally (3) R satisfies one or more properties.

The crucial upgrade: replacing the ZK proof with a PBS. We modify the last stage of DECO, deviating from the design in [1] and embedding the proof generation phase directly within a blind signature protocol. Note that, in Fig. 1, the "BlindSignature Protocol" in the dashed box shows our protocol's deviation from [1]. In particular, after having verified locally the integrity of the response R received from $\mathcal{S}$, $\mathcal{P}$ hides R by running with some randomness rand a probabilistic algorithm, $\mathsf{Blind}(\cdot)$, which hides R inside another message R' given in output. R' is sent to $\mathcal{N}$ by $\mathcal{P}$. Then, in the proof generation phase, $\mathcal{P}$ additionally proves to $\mathcal{N}$ that 4) she knows the message R and the randomness rand that have been used as input of $\mathsf{Blind}(\cdot)$ to produce R'. Note that, to ensure that R actually comes from $\mathcal{S}$, $\mathcal{P}$ slightly modifies R adding a piece of information stating that R comes from a TLS connection with $\mathcal{S}$ and proving to $\mathcal{N}$ that only this modification has been added to R (to avoid burdening the notation, from now on we refer also to this information when using R). If the proof is verified correctly, the notary $\mathcal{N}$ produces a signature σ' on R'. Once received σ', $\mathcal{P}$ runs another algorithm $\mathsf{Unblind}(\cdot)$ that takes as input[4] σ' and rand and that unblindsthe signature σ' producing a new signature σ that is verified correctly on R. In the end, σ is a cryptographic signature produced by $\mathcal{N}$ without accessing the message R, thanks to the blindness property of the PBS scheme. The prover $\mathcal{P}$ can now present σ and R to any verifier $\mathcal{V}$ to convince her that $\mathcal{P}$ has downloaded R from a server $\mathcal{S}$. Indeed, if σ is correctly verified with $\mathsf{pk}_\mathcal{N}$, $\mathcal{V}$ trusts that $\mathcal{N}$ has audited the communication between $\mathcal{P}$ and $\mathcal{S}$.

Note that when the response R is not randomized, $\mathcal{P}$ can repeat the protocol summarized in Fig. 1 obtaining the same response R multiple times, involving different notaries, to obtain a set of signatures that can be combined into a single multisignature. In this way, we mitigate the risks associated to a collusion between $\mathcal{P}$ and $\mathcal{N}$.

[4]A run of the (un)blinding takes as input also the public key of $\mathcal{N}$.

Threat model. The threat model along with the setting and the actor we consider is in Fig. 1. In our system, we consider four parties: the server $\mathcal{S}$, the user (also referred to as the prover following DECO's notation) $\mathcal{P}$, the notary $\mathcal{N}$, and the verifier $\mathcal{V}$. The user $\mathcal{P}$ wishes to obtain authenticated data from the server $\mathcal{S}$. The notary $\mathcal{N}$ helps in auditing the TLS communication but without learning the context of such communication (i.e., in a way that $\mathcal{P}$ preserves privacy of its obtained data). Finally, the verifier $\mathcal{V}$ is a party that interacts solely with $\mathcal{P}$, who seeks to convince $\mathcal{V}$ that the data received from $\mathcal{S}$ has been certified by the notary $\mathcal{N}$.

Our threat model follows that of DECO, with modifications to account for our setting. In particular, we consider the following adversarial capabilities: (1) A potentially malicious user $\mathcal{P}$ may attempt to convince the verifier $\mathcal{V}$ that self-computed data originated from the server $\mathcal{S}$, thereby fooling the notary $\mathcal{N}$ into attesting to self-computed data. (2) A potentially malicious notary $\mathcal{N}$ may attempt to extract confidential information from the user's data while auditing the interaction with the server $\mathcal{S}$. (3) Importantly, we also consider collusion between $\mathcal{P}$ and $\mathcal{N}$, wherein they cooperate to convince the verifier $\mathcal{V}$ of the authenticity of data that was not generated by the server $\mathcal{S}$. In such cases, $\mathcal{N}$ may attest to data crafted by $\mathcal{P}$ without any legitimate interaction with $\mathcal{S}$.

Our contribution: point-by-point. We summarize the main contributions of our work below.

► We design a Predicate Blind Signature (PBS) protocol that produces standard RSA-PSS signatures, preserving compatibility with widely adopted software (e.g., Adobe Acrobat Reader). Our construction extends the RSA-BSSA scheme defined by the IETF standard, enforcing the verification of a predicate, thus allowing a notary to blindly sign hidden data only if it satisfies the agreed predicate. This required a significantly different design and security analysis, both of which are key parts of our contribution. The protocol is two-round and makes use of a non-interactive zero-knowledge (NIZK) proof to ensure the correctness of the predicate without revealing the message. The resulting protocol satisfies *blindness*, based on the ZK property of the underlying NIZK, and also satisfies *one-more unforgeability*, relying on the unforgeability of RSA-PSS and the proof-of-knowledge property of the NIZK.

► We define a predicate that certifies that some data were correctly retrieved from a TLS server, leveraging the DECO architecture: in particular, we design the predicate to guarantee to the signer that the ciphertexts of the web requests and responses were correctly formed and authenticated, and that the plaintext response satisfies additional application-specific conditions. We consider this within a four-party setting that involves a prover, a server, a notary, and a verifier that, with our PBS protocol, has to rely only on legacy signature verification.

► We implement and evaluate the entire construction in a realistic TLS scenario, where a user owning a PDF file of a realistic size (i.e., 26 KB) obtains a predicate blind RSA-

PSS signature from the notary. The resulting PDF is verifiable with standard software such as Adobe Acrobat Reader, without requiring any non-standard cryptographic tool.

► We implement the NIZK proof system in Circom and evaluate the performance of two frameworks: (1) a VOLE-in-the-head (VitH) proof system offering scalability and low memory usage; and (2) Groth16 SNARKs, offering succinct proofs and fast verification. We demonstrate that the most demanding part (generating the proof that the blinded message corresponds to data encrypted with AES) can be performed efficiently, and we show how to scale the construction to large files by chunking the SHA256 and AES-CTR computation, allowing proof generation even on low-resource devices.

II. Preliminaries

We refer to App. A for standard definitions of cryptographic primitives (e.g., RSA-PSS) and notations (which are almost identical to those used in [12]).

A. Definition of Predicate Blind Signature Schemes

Here we provide the definition of (predicate) blind signatures (PBS) according to the security model presented in [12]. In [12] introduces the concept of PBS, which is a generalization of standard and partial blind signatures [20]. A PBS scheme is an interactive protocol that enables a signer to sign a message for another party, called the user, without learning anything about the signed message, except that it satisfies certain conditions (defined by a predicate) on which the user and signer agreed before the interaction.

A PBS scheme is parameterized by a family of polynomial time computable predicates, which are implemented by a polynomial time algorithm $\mathsf{P} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$, the predicate compiler, that on input a predicate description $\varphi \in \{0,1\}^*$ and a message $m \in \{0,1\}^*$, outputs 1 if m satisfies φ, otherwise 0. In a PBS scheme $\mathsf{PBS[P]}$ for P we have the following algorithms[5].

- $\mathsf{PBS.Stp}(1^\lambda) \to \mathsf{par}$ on input λ, outputs public parameters par, which define a message space $\mathcal{M}_{\mathsf{PBS}}$.
- $\mathsf{PBS.KG}(\mathsf{par}) \to (\mathsf{sk}, \mathsf{vk})$ on input the parameters par, outputs a signing/verification key pair $(\mathsf{sk}, \mathsf{vk})$, implicitly containing par, namely $\mathsf{par} :\subseteq \mathsf{vk}$.
- $\langle \mathsf{PBS.S}(\mathsf{sk}, \varphi), \mathsf{PBS.U}(\mathsf{vk}, \varphi, m)\rangle \to (0/1, \sigma/\perp)$ is an *interactive protocol*, with shared input par (implicit in sk and vk) and a predicate φ, that is run between two PPT algorithms $\mathsf{PBS.S}$, the signer algorithm (or the signer) and $\mathsf{PBS.U}$, the user algorithm (or the user). The signer takes a signing key sk as private input, the user's private input is a verification key vk and a message m. The signer outputs 1 if the interaction completes successfully and 0 otherwise, while the user outputs a signature σ if it terminates correctly, and $\perp$ otherwise.
  Note that $\mathsf{PBS.U}$ is composed by two sub-algorithms, namely $\mathsf{PBS.U} = (\mathsf{PBS.UBld}, \mathsf{PBS.UFin})$ and $\mathsf{PBS.S}$ defines the algorithm $\mathsf{PBS.SSig}$. The interaction, with b representing a bit, is defined as follows:
$$\begin{aligned}&(\mathsf{msg}_0, \mathsf{st}) \leftarrow \mathsf{PBS.UBld}(\mathsf{vk}, \varphi, m),\\ &(\mathsf{msg}_1, b) \leftarrow \mathsf{PBS.SSig}(\mathsf{sk}, \varphi, \mathsf{msg}_0),\\ &\sigma \leftarrow \mathsf{PBS.UFin}(\mathsf{st}, \mathsf{msg}_1)\end{aligned}$$
  As shorthand for the above sequence we write $(0/1, \sigma/\perp) \leftarrow \langle \mathsf{PBS.S}(\mathsf{sk}, \varphi), \mathsf{PBS.U}(\mathsf{vk}, \varphi, m)\rangle$.
- $\mathsf{PBS.Vry}(\mathsf{vk}, m, \sigma) \to 0/1$ is deterministic and on input a verification key vk, a message m, and a signature σ, outputs 1 if σ is valid on m under vk and 0 otherwise.

and such that Correctness, One-More Unforgeability and Blindness as defined below hold.

Definition 1 (Correctness)**.** *A predicate blind signature scheme* PBS *for predicate compiler* P *is perfectly correct if for every* PPT *adversary* $\mathcal{A}$ *and* $\lambda \in \mathbb{N}$ *the probability in (1) is equal to* 1.

a) One-More Unforgeability.: The one-more unforgeability (OMUF) property states that after the completion of ℓ signing sessions, the user cannot compute $\ell + 1$ distinct valid message-signature pairs.

In [12] this notion is generalized for PBS schemes. Loosely speaking, the OMUF property for PBS requires that any output generated by the user after participating in signing sessions, using predicates of its choice, must be explicable with respect to a predicate that was actually used during the sessions. Specifically, let ℓ be the number of closed signing sessions, and let φ_j be the predicate used in the j-th session. Suppose that the message-signature pairs obtained by the adversary are $(m_i^*, \sigma_i^*)_{i\in[\kappa]}$. The OMUF property requires the existence of an *injective mapping* $f : [\kappa] \to [\ell]$ so that $\mathsf{P}(\varphi_{f(k)}, m_k^*) = 1$ for all $k \in [\kappa]$.

Definition 2 (OMUF)**.** *Consider the game* OMUF *in Fig. 2. A predicate blind signature scheme* PBS *for a predicate compiler* P *satisfies one-more unforgeability (OMUF) if for every adversary* $\mathcal{A}$ *the advantage* $\mathsf{Adv}^{\mathrm{omuf}}_{\mathsf{PBS[P]}}(\mathcal{A}, \lambda)$ *is negligible in* λ *where:* $\mathsf{Adv}^{\mathrm{omuf}}_{\mathsf{PBS[P]}}(\mathcal{A}, \lambda) := \Pr\Big[\mathrm{OMUF}^{\mathcal{A}}_{\mathsf{PBS[P]}}(\lambda) = 1\Big]$.

b) Blindness.: We recall here the notion of blindness for PBS schemes as in [12]. Following their approach, we adopt the same definition of blindness for schemes with parameters, which also covers instantiations without parameters (or with "empty" parameters). Loosely speaking, the blindness property for PBS requires that if a signer and a user (or multiple users) interact n times in n different sessions, producing n signatures, then later, when the signer sees one of these n signatures (along with the message for that signature), it cannot identify (with more than negligible probability) in which session it was issued. The definition is given in the (more demanding) *malicious-signer model* [22].

Definition 3 (Blindness)**.** *Consider the game* BLD *in Fig. 3. A predicate blind signature scheme* PBS *for a predicate compiler* P *satisfies blindness if for every adversary* $\mathcal{A}$ *the advantage*

[5]Note that here following previous works (e.g., see [21, 12]) on blind signature, we give a definition for a 2-round PBS scheme.

$$\Pr\left[\begin{array}{c|c} m \notin \mathcal{M}_{\mathsf{PBS}} \vee & \mathsf{par} \leftarrow \mathsf{PBS.Stp}(1^\lambda), (\mathsf{sk},\mathsf{vk}) \leftarrow \mathsf{PBS.KG}(\mathsf{par}), \\ \mathsf{P}(\varphi, m) = 0 \vee & (m,\varphi) \leftarrow \mathcal{A}(\mathsf{sk},\mathsf{vk}), \\ (b \wedge b') & (b,\sigma) \leftarrow \langle \mathsf{PBS.S}(\mathsf{sk},\varphi), \mathsf{PBS.U}(\mathsf{vk},\varphi,m)\rangle, \\ & b' := \mathsf{PBS.Vry}(\mathsf{vk}, m, \sigma) \end{array}\right] \tag{1}$$

```
Game OMUF$^{\mathcal{A}}_{\mathsf{PBS[P]}}(\lambda)$
1 : $\mathsf{par} \leftarrow \mathsf{PBS.Stp}(1^\lambda)$
2 : $(\mathsf{sk},\mathsf{vk}) \leftarrow \mathsf{PBS.KG}(\mathsf{par}); \mathbf{P}{:=}[]$
3 : $(m_i^*, \sigma_i^*)_{i\in[\ell]} \leftarrow \mathcal{A}^{\mathsf{Sign}}(\mathsf{vk})$
4 : if $\exists i_1 \neq i_2 : m^*_{i_1} = m^*_{i_2}$ return $0$
5 : if $\exists i \in [\ell] : \mathsf{PBS.Vry}(\mathsf{vk}, m_i^*, \sigma_i^*) \neq 1$ return $0$
6 : if $\exists f \in \mathcal{F}([\ell], [|\mathbf{P}|]) : \forall i \in [\ell] : \mathsf{P}(\mathbf{P}_{f(i)}, m_i^*) = 1$ return $0$
7 : return $1$

Oracle Sign$(\varphi, \mathsf{msg_{in}})$
1 : $(\mathsf{msg_{out}}, b) \leftarrow \mathsf{PBS.SSig}(\mathsf{sk}, \varphi, \mathsf{msg_{in}})$
2 : if $b \neq 1$ return $\mathsf{msg_{out}}$
3 : $\mathbf{P} = \mathbf{P} \leftarrow \mathbf{P}\|\varphi$
4 : return $\mathsf{msg_{out}}$
```

Fig. 2: (**Definition**) The OMUF game for a PBS $\mathsf{PBS[P]}$. $\mathcal{F}(\mathcal{I}, \mathcal{J})$ is the *set* of injective functions from set $\mathcal{I}$ to set $\mathcal{J}$.

$\mathsf{Adv}^{\mathrm{bld}}_{\mathsf{PBS[P]}}(\mathcal{A},\lambda)$ *is negligible in* λ *where:* $\mathsf{Adv}^{\mathrm{bld}}_{\mathsf{PBS[P]}}(\mathcal{A},\lambda) := \left|\Pr\left[\mathrm{BLD}^{\mathcal{A},1}_{\mathsf{PBS[P]}}(\lambda) = 1\right] - \Pr\left[\mathrm{BLD}^{\mathcal{A},0}_{\mathsf{PBS[P]}}(\lambda) = 1\right]\right|$.

In Fig. 3 sess_i for $i \in \{0,1\}$ indicates the round of the i-th session of the protocol. Moreover, we highlight that in the Finalize oracle, if sess_{1-i} is not equal to `closed` the output is $(i, \texttt{closed})$. Note that the blindness notions of [12], which follow the classical definitions from previous works, can only ensure the user's privacy if, at the time the user publishes a signature, the signer has blindly signed a sufficiently large number of messages under the same key. In the case of predicate blind signatures, this requires that many message predicates are satisfied by the signed message. Moreover, in the BLD game in Fig. 3, by allowing the adversary $\mathcal{A}_1$ to output distinct predicates φ_0, φ_1, the blindness notion also ensures that the resulting signature does not reveal any information about the predicate that was used.

III. Construction of Predicate RSA-PSS Blind Signature

We give a construction based on "plain" blind RSA-PSS signatures, called also RSA-BSSA [23]. In a nutshell, the protocol is 2-round where it first speaks the user PBS.UBld and then it responds the signer PBS.SSig algorithm, finally the user finalize the signature by using the algorithm PBS.UFin. To obtain predicate blind signature, in the first round we require the user to send a proof that will assert that the "blinded" message that it is sending satisfies the agreed-upon predicate φ. We, therefore, first introduce the following relation $\mathsf{R_{Blind}}$:

```
$\mathsf{R_{Blind}}(\ \mathbb{x}{:=}(N, e, \mathsf{H_{PSS}}, t), \mathbb{w}{:=}(m, r, \kappa)\ )$ :
$\mu{:=}\mathsf{H_{PSS}.Enc}(m, r)$
return $(\mathsf{P}(\varphi, m) = 1 \wedge \mu\kappa^e \equiv t \pmod N)$
```

This relation $\mathsf{R_{Blind}}$ checks whether the user's message t was blinded starting from an hidden message m and a randomness κ according to the public pair (N, e), and that m satisfies the predicate φ.

Let RSAGen be an RSA key generation algorithm and PSSGen a PSS function generator (which together define RSA.KG in Fig. 8), let P be a predicate compiler for which the relation $\mathsf{R_{Blind}}$ is defined, and let PS be a NIZK argument system for $\mathsf{R_{Blind}}$; we describe in Fig. 4 the 2-round PBS scheme $\mathsf{PBS_{RSA}}[\mathsf{P}, \mathsf{RSAGen}, \mathsf{PSSGen}, \mathsf{PS}]$.

In a nutshell, our PBS scheme follows RSA-PSS (and its blind version in [23]). The key generation algorithm $\mathsf{PBS_{RSA}.KG}$ generates an RSA key pair $\mathsf{vk}{:=}(N, e, \mathsf{H_{PSS}}, \mathsf{par})$, $\mathsf{sk}{:=}(N, e, d, \mathsf{H_{PSS}}, \mathsf{par})$, where $ed \equiv 1 \pmod{\phi(N)}$. Following Chaum, to obtain a blind signature on a message m, the user first generates a PSS encoding $\mu \leftarrow \mathsf{H_{PSS}.Enc}(m, r)$ with $r \leftarrow\!\$\ \{0,1\}^\lambda$, then blinds it using a random $\kappa \leftarrow\!\$\ [N]$ obtaining $t = \mu\kappa^e \pmod N$, which is an element of $\mathbb{Z}_N^*$ that is distributed independently of m. Then he gets from the signer the blinded signature $\mathsf{msg}_1{:=}t^d \pmod N$, and unblinds it to obtain and output $\sigma{:=}\mathsf{msg}_1\kappa^{-1} \pmod N$. To verify a signature σ on a message m, follow the same algorithm as RSA-PSS verification.

Note that, to be a PBS scheme, the user, along with t, also sends a NIZK argument of knowledge π proving that the message m (that is in t after being encoded and blinded) satisfies some predicate φ shared with the signer. That is, the signer before sending msg_1 to the user (and enabling him to compute the signature σ) first checks if the proof π is correct and in case the verification fails, it aborts.

We provide a schematic overview of the interaction:

Signer(sk, φ)	User$(\mathsf{vk}, \varphi, m)$
	$((t,\pi),\kappa) \leftarrow \mathsf{PBS_{RSA}.UBld}(\mathsf{vk}, \varphi, m)$
$(t^d \pmod N, b) \leftarrow \mathsf{PBS_{RSA}.SSig}(\mathsf{sk}, \varphi, (t,\pi))$	
	$\sigma \leftarrow \mathsf{PBS_{RSA}.UFin}(\kappa, t^d \pmod N)$

Correctness. The perfect correctness follows from the perfect correctness of PS scheme (see Def. 9) and the unforgeability of RSA (see Assumption 1). We show now that the scheme

```
Game $\mathrm{BLD}^{\mathcal{A},b}_{\mathsf{PBS[P]}}(\lambda)$
$\mathsf{par} \leftarrow \mathsf{PBS.Stp}(1^\lambda); b_0{:=}b; b_1{:=}(1-b)$
$(\varphi_0, \varphi_1, m_0, m_1, key, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{par})$
if $\exists i,j \in \{0,1\} : \mathsf{P}(\varphi_i, m_j) \neq 1$
 return $0$
$(\mathsf{sess}_0, \mathsf{sess}_1){:=}(\mathtt{init}, \mathtt{init})$
$b' \leftarrow \mathcal{A}_2^{\mathsf{Blind},\mathsf{Finalize}}(\mathsf{st})$
return $(b = b')$
```

```
Oracle $\mathsf{Blind}(i)$
if $i \notin \{0,1\} \vee \mathsf{sess}_i \neq \mathtt{init}$
    return $\perp$
$\mathsf{sess}_i = \mathtt{open}; \mathsf{vk}{:=}(\mathsf{par}, key)$
$(\mathsf{msg}, \mathsf{st}_i) \leftarrow \mathsf{PBS.UBld}(\mathsf{vk}, \varphi_i, m_{b_i})$
return $\mathsf{msg}$
```

```
Oracle $\mathsf{Finalize}(i, \mathsf{msg})$
if $i \notin \{0,1\} \vee \mathsf{sess}_i \neq \mathtt{open}$ return $\perp$
$\mathsf{sess}_i = \mathtt{closed}$
$\sigma_{b_i} \leftarrow \mathsf{PBS.UFin}(\mathsf{st}_i, \mathsf{msg})$
if $\mathsf{sess}_0 = \mathsf{sess}_1 = \mathtt{closed}$
    if $\sigma_0 = \perp \vee \sigma_1 = \perp$ return $(\perp, \perp)$
    return $(\sigma_0, \sigma_1)$
return $(i, \mathtt{closed})$
```

Fig. 3: (**Definition**) The BLD game for a PBS scheme $\mathsf{PBS[P]}$ with an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$.

```
Alg. $\mathsf{PBS_{RSA}.Stp}(1^\lambda)$
$(\mathsf{crs}, \tau) \leftarrow \mathsf{PS.Stp}(1^\lambda)$
return $\mathsf{par}{:=}\mathsf{crs}$
```

```
Alg. $\mathsf{PBS_{RSA}.KG}(\mathsf{par})$
$(N, e, d) \leftarrow \mathsf{RSAGen}(1^\lambda)$
$\mathsf{H_{PSS}} \leftarrow \mathsf{PSSGen}(N)$
$\mathsf{sk}{:=}(N, e, d, \mathsf{H_{PSS}}, \mathsf{par})$
$\mathsf{vk}{:=}(N, e, \mathsf{H_{PSS}}, \mathsf{par})$
return $(\mathsf{sk}, \mathsf{vk})$
```

```
Alg. $\mathsf{PBS_{RSA}.Vry}(\mathsf{vk}, m, \sigma)$
$(N, e, \mathsf{H_{PSS}}) :\subseteq \mathsf{vk}$
$\mu'{:=}\sigma^e (\mathrm{mod}\ N)$
$m' \leftarrow \mathsf{H_{PSS}.Dec}(\mu')$
return $(m' = m)$
```

```
Alg. $\mathsf{PBS_{RSA}.UBld}(\mathsf{vk}, \varphi, m)$
$(N, e, \mathsf{H_{PSS}}, \mathsf{crs}) :\subseteq \mathsf{vk}; r \leftarrow\!\!\$\ \{0,1\}^\lambda$
$\kappa \leftarrow\!\!\$\ [N]; \mu \leftarrow \mathsf{H_{PSS}.Enc}(m, r)$
$t{:=}\mu\kappa^e (\mathrm{mod}\ N)$
$\mathbb{x}{:=}(N, e, \mathsf{H_{PSS}}, t); \mathbb{w}{:=}(m, r, \kappa)$
$\pi \leftarrow \mathsf{PS.Prv}(\mathsf{crs}, \mathbb{x}, \mathbb{w})$
$\mathsf{msg}_0{:=}(t, \pi), \mathsf{st}{:=}\kappa$
return $(\mathsf{msg}_0, \mathsf{st})$
```

```
Alg. $\mathsf{PBS_{RSA}.SSig}(\mathsf{sk}, \varphi, \mathsf{msg}_0)$
$(t, \pi){:=}\mathsf{msg}_0$
$(N, e, d, \mathsf{H_{PSS}}, \mathsf{crs}) :\subseteq \mathsf{sk}$
$\mathbb{x}{:=}(N, e, \mathsf{H_{PSS}}, t)$
if $\mathsf{PS.Vry}(\mathsf{crs}, \mathbb{x}, \pi) \neq 1$
    return $(\perp, 0)$
$\mathsf{msg}_1{:=}t^d (\mathrm{mod}\ N)$
return $(\mathsf{msg}_1, 1)$
```

```
Alg. $\mathsf{PBS_{RSA}.UFin}(\mathsf{st}, \mathsf{msg}_1)$
$\kappa{:=}\mathsf{st}$
$\sigma{:=}\mathsf{msg}_1\kappa^{-1} (\mathrm{mod}\ N)$
return $\sigma$
```

Fig. 4: (**Construction**) $\mathsf{PBS_{RSA}[P, RSAGen, PSSGen, PS]}$ on predicate compiler P, an RSA key generation algorithm RSAGen, a PSS function generator PSSGen according to Def. 8, and a NIZK argument PS for the relation $\mathsf{R_{Blind}}$.

presented in Fig. 4 satisfies the properties of OMUF and Blindness according to Defs. 2 and 3.

Theorem 1 (Blindness)**.** *Let* P *be a predicate compiler,* RSAGen *and* PSSGen *be an RSA key and a PSS function generation algorithms; let* $\mathsf{PS[R_{Blind}]}$ *be a NIZK argument of knowledge for the relation* $\mathsf{R_{Blind}}$*. Then, for any adversary* $\mathcal{A}$ *playing in the game* BLD *against the PBS scheme* $\mathsf{PBS_{RSA}[P, RSAGen, PSSGen, PS]}$ *defined in Fig. 4, there exists algorithms* $\mathcal{Z}_0$ *and* $\mathcal{Z}_1$*, playing in game* ZK *against zero-knowledge of* $\mathsf{PS[R_{Blind}]}$ *such that for every* $\lambda \in \mathbb{N}$*:* $\mathsf{Adv}^{\mathrm{bld}}_{\mathsf{PBS_{RSA}}}(\mathcal{A}, \lambda) \leq \mathsf{Adv}^{\mathrm{zk}}_{\mathsf{PS[R_{Blind}]}}(\mathcal{Z}_0, \lambda) + \mathsf{Adv}^{\mathrm{zk}}_{\mathsf{PS[R_{Blind}]}}(\mathcal{Z}_1, \lambda)$.

The idea behind the proof of Thm. 1 is the following. The proof proceeds via hybrid games. That is, starting with (real) game $\mathrm{BLD}^{\mathcal{A},b}_{\mathsf{PBS_{RSA}}}$ for an arbitrarily fixed b, we replace the user's proofs π (in both signing sessions) by simulated proofs. The hybrid and the real game are indistinguishable by the zero knowledge of $\mathsf{PS[R_{Blind}]}$. Now using a similar argument for "plain" blind RSA, this final game is independent of the bit b, which concludes the proof. The formal proof is in App. C.

Theorem 2 (OMUF)**.** *Let* P *be a predicate compiler,* RSAGen *and* PSSGen *be an RSA key and a PSS function generation algorithms; let* $\mathsf{PS[R_{Blind}]}$ *be a NIZK argument of knowledge for the relation* $\mathsf{R_{Blind}}$*, and let* $\mathsf{RSA[RSAGen, PSSGen]}$ *be the RSA signature scheme of Fig. 8 instantiated with* RSAGen *and* PSSGen*. Then, for any adversary* $\mathcal{A}$ *playing in the game* OMUF *against the PBS scheme* $\mathsf{PBS_{RSA}[P, RSAGen, PSSGen, PS]}$ *defined in Fig. 4, there exists algorithms* $\mathcal{P}$ *playing in game* POK *against proof of knowledge of* $\mathsf{PS[R_{Blind}]}$ *and* $\mathcal{R}$ *playing in game* EUF-CMA *against the unforgeability of* $\mathsf{RSA[RSAGen, PSSGen]}$ *such that for every* $\lambda \in \mathbb{N}$*:* $\mathsf{Adv}^{\mathrm{omuf}}_{\mathsf{PBS_{RSA}}}(\mathcal{A}, \lambda) \leq \mathsf{Adv}^{\mathrm{pok}}_{\mathsf{PS[R_{Blind}]}}(\mathcal{P}, \lambda) + \mathsf{Adv}^{\mathrm{euf\text{-}cma}}_{\mathsf{RSA[RSAGen,PSSGen]}}(\mathcal{R}, \lambda)$.

We bound the advantage in breaking the unforgeability of $\mathsf{PBS_{RSA}}$, according to Def. 2, by the advantages in breaking the security of underlying primitives. In Assumption 1 we directly assume EUF-CMA security of the RSA-PSS [10] signature scheme. The reason is that all known security proofs of RSA-PSS signatures are in the random-oracle model, but the PS relation $\mathsf{R_{Blind}}$ depends on the used hash function, which would be replaced in the analysis by a random function, for which efficient proofs are not possible.

Following the same arguments as in [12] (which are even stronger in this case, given the widespread use of RSA-PSS in commercial products as introduced before), Assumption 1 might be unconventional from a theoretical perspective, but is arguably uncontroversial in practice due to the extensive use of RSA-PSS signatures. In any case, such an assumption is inherent to any application involving RSA signatures.

The idea behind the proof of Thm. 2 is to reduce the unforgeability of $\mathsf{PBS}_{\mathsf{RSA}}$ to the unforgeability of RSA-PSS signatures. Given a verification key $\mathsf{vk} = (N, e, \mathsf{H}_{\mathsf{PSS}})$, the reduction sets up the crs and, after the setup, provides the verification key for the $\mathsf{PBS}_{\mathsf{RSA}}$ scheme to the adversary. It then answers the adversary's signing queries to the oracle Sign according to the OMUF game in Fig. 2. That is, when the adversary queries the oracle Sign with input message (t, π), as described in Fig. 4, the reduction uses the trapdoor corresponding to crs to extract (m, r, κ) (i.e., the witness used to generate π). It then queries its own signing oracle for a signature on m and returns to the adversary the product of this signature and κ, which is exactly what the adversary expects. The proof proceeds formally via a sequence of hybrid games, described in App. C.

A. $\mathsf{PBS}_{\mathsf{RSA}}$ *to Certify Data in DECO*

The construction $\mathsf{PBS}_{\mathsf{RSA}}$ in Fig. 4 blindly outputs an RSA-PSS signature on a message satisfying a generic predicate φ. In this section, we explain how to instantiate the construction in Fig. 4 in the specific case of DECO [1]. That is, we focus on identifying the predicate φ when $\mathsf{PBS}_{\mathsf{RSA}}$ is used within DECO, yielding the concrete realization of **ACTS**.

Before doing so, we introduce some additional notation regarding authenticated encryption schemes. An authenticated encryption scheme is defined as a tuple of efficient algorithms $\mathsf{AE} = (\mathsf{AE.KG}, \mathsf{AE.Enc}, \mathsf{AE.Dec})$, where $\mathsf{AE.KG}(1^\lambda) \to \mathsf{k}$ is a key generation algorithm that, on input the security parameter, outputs a symmetric encryption and authentication key k; $\mathsf{AE.Enc}_\mathsf{k}(m) \to (c, t)$ is an encryption algorithm that, on input a key k and a plaintext m, outputs a ciphertext c and an authentication tag t; and $\mathsf{AE.Dec}_\mathsf{k}(c, t) \to m'$ is a decryption algorithm that, on input a key k, a ciphertext c, and an authentication tag t, returns either the plaintext m' or a failure symbol. For AE's security, see Definition 5.4 in [24]; for a standard encrypt-then-authenticate construction (also valid for DECO), see Construction 5.6 in [24].

We remind the reader that, in this phase, we are in a setting (as in DECO [1]) with three parties: a notary $\mathcal{N}$, a prover $\mathcal{P}$ and a server $\mathcal{S}$. The predicate blind signature protocol is between $\mathcal{P}$ and $\mathcal{N}$ and during this protocol $\mathcal{P}$ proves to $\mathcal{N}$ that the message sent to $\mathcal{S}$ are formed according to a shared structure (i.e., a template $\mathsf{Query}(\cdot)$ that takes as input a string θ_S known only by $\mathcal{P}$), that the message R received by $\mathcal{S}$ from $\mathcal{P}$ are authentic and that R respects some properties. For some applications, it is relevant only that $\mathcal{P}$ has been able to download R from $\mathcal{S}$ without checking whether the query to $\mathcal{S}$ was well formed. However, for completeness, in the following we will check also that the queries are correctly built.

Following Section 4.3 of [12], we generalize predicates to NP-relations, letting P take as input φ, m, and a witness w such that $\mathsf{P}(\varphi, m, \mathsf{w}) = 1$, which is equivalent to the formulation in Sec. II-A. Then, we construct m, φ and w in the following way: m is composed by R; w is composed by a string θ_S, the authentication tags R_{tag} and Q_{tag} and share of an encryption secret key $\mathbf{k}_{\mathcal{P}}$; φ is composed by the ciphertexts R_{enc} and Q_{enc}, a share of an encryption secret key $\mathbf{k}_{\mathcal{N}}$ and a template of a query $\mathsf{Query}(\cdot)$.

The predicate compiler P, first, recomputes k' from the shares $\mathbf{k}_{\mathcal{P}}$ and $\mathbf{k}_{\mathcal{N}}$, and then returns 1 if and only if $\mathsf{AE.Dec}_{\mathbf{k}'}(R_{enc}, R_{tag}) = R$ and $\mathsf{AE.Dec}_{\mathbf{k}'}(Q_{enc}, Q_{tag}) = \mathsf{Query}(\theta_S)$. Note that P can verify an additional predicate on R, but we omit it to simplify presentation.

We highlight that μ of $\mathsf{R}_{\mathsf{Blind}}$ is computed only from m.

Overview of interactions. We now provide an overview of the interactions of our construction between the notary and the prover in the specific case of DECO. To better visualize it, we suggest to the reader to refer to Fig. 1.

As discussed above, once the communication with $\mathcal{S}$ ends, the prover commits to Q and R. It does so by sharing with the notary Q_{enc} and R_{enc}. In Fig. 1, this is highlighted with the message labeled "Commit to (Q,R)".

After receiving $\mathbf{k}_{\mathcal{N}}$ from $\mathcal{N}$, the prover runs $\mathsf{PBS}_{\mathsf{RSA}}.\mathsf{UBld}(\mathsf{vk}, (R_{enc}, Q_{enc}, \mathbf{k}_{\mathcal{N}}, \mathsf{Query}(\cdot)), R, (\theta_S, R_{tag}, Q_{tag}, \mathbf{k}_{\mathcal{P}}))$ obtaining t and the proof π which are then sent to $\mathcal{N}$ (note that, similar to what was done in Section 4.3 of [12], here we extend the definition of $\mathsf{PBS}_{\mathsf{RSA}}.\mathsf{UBld}$ taking the additional argument w). In Fig. 1, this corresponds to the step labeled in red as "Compute $(R', \mathsf{rand}) \leftarrow \mathsf{Blind}(R)$" and the box "Proof Generation".

The notary then runs $\mathsf{PBS}_{\mathsf{RSA}}.\mathsf{SSig}(\mathsf{sk}, (R_{enc}, Q_{enc}, \mathbf{k}_{\mathcal{N}}, \mathsf{Query}(\cdot)), (t, \pi))$ obtaining the blinded signature $\mathsf{msg}_1 := t^d \pmod N$ (which in Fig. 1 corresponds to σ') and sends msg_1 with $\mathcal{P}$.

Finally, $\mathcal{P}$ runs $\mathsf{PBS}_{\mathsf{RSA}}.\mathsf{UFin}(\kappa, \mathsf{msg}_1)$ obtaining the signature σ. In Fig. 1, this corresponds to the step labeled in red as "Compute $\sigma \leftarrow \mathsf{Unblind}(\sigma', \mathsf{rand})$".

Security of the full protocol. DECO's theoretical security is proven in the UC framework, showing that the protocol securely realizes a DECO ideal functionality in a hybrid model that assumes access to both secure 2PC computation and ZK functionalities (see Theorem 4.1 in [1]).

We can use the very same approach to provide similar security guarantees (for our DECO's variant) by replacing the ZK functionality with Fischlin's blind signature functionality [25][6]. That is, we keep the rest of the DECO protocol mostly unchanged, and only modify the last phase to use predicate blind signatures instead of NIZKs. By Canetti's composition theorem [26], this change preserves the overall security, and the proof is straightforward.

[6]More precisely, we require the natural extension of the blind signature functionality in [25] to predicate blind signatures.

Nevertheless, we note that, while DECO's security proof is based on ZK ideal functionality, their implementation uses zkSNARKs [27], which do not come with proven composition guarantees. This creates a clear gap between the security proof and the actual deployed system since no composition theorem of Canetti can be used to achieve security when such a zkSNARK is replaced with the ZK functionality.

We prefer to focus on the concrete security of the $\mathsf{PBS}_{\mathsf{RSA}}$ component under (self) composition. We also support this with experimental results that demonstrate its efficiency. This gives a better match between the theory and the implementation, and in our view, leads to more meaningful security guarantees. Although it is possible to state and prove a theorem similar to Theorem 4.1 in DECO (mainly using the composition theorem), we do not include it, since the actual implementation cannot be replaced by the functionality (as in DECO's case).

B. Extending to Predicate Blind Multisignature schemes for facing malicious coalitions in DECO.

As we have already discussed, there can be real-world scenarios where the user and notary might decide to collude, especially in applications in which the content exchanged with the server has a relevant value and the user can therefore try to bribe the notary.

We can strongly mitigate the above risk by forcing a prover to run $\mathsf{PBS}_{\mathsf{RSA}}$ with multiple distinct notaries. In this way, we construct a predicate blind multisignatures. In the following, we will explain how to instantiate such a construction.

Before doing so, note that a blind multisignature scheme has recently been defined in [13] (Sec. 3). We can easily extend this scheme into a predicate blind multisignature one PBMS by: parameterizing the definition of [13] by a family of computable polynomial-time predicates (as shown in Sec. II-A); adding a setup algorithm $\mathsf{PBMS.Stp}$ that on input λ outputs public parameters par; substituting the interactive protocol between the user and a signer outputting a signature σ ($\mathsf{BMS.Sign}\langle\cdot\rangle$) with an interactive protocol in which every parties also take a predicate description as input (as shown in Sec. II-A); changing the game of the one-more unforgeability property (Def. 2 of [13]) allowing the adversary to access the Sign oracle of Fig. 2 instead of the signing oracle of Def. 2 of [13]; changing the game of the blindness property (Def. 3 of [13]) allowing the adversary to output two predicate description φ_0, φ_1 and access the Blind and $\mathsf{Finalize}$ during the game. In the following, we denote the algorithms of PBMS the same way as it is done in [13] (Sec. 3).

Now, we give an informal construction $\mathsf{PBMS}_{\mathsf{RSA}}$ of a predicate blind multisignature based on $\mathsf{PBS}_{\mathsf{RSA}}$: (**1**) The setup algorithm $\mathsf{PBMS}_{\mathsf{RSA}}.\mathsf{Stp}$ runs $\mathsf{PBS}_{\mathsf{RSA}}.\mathsf{Stp}$ and outputs par obtained from $\mathsf{PBS}_{\mathsf{RSA}}.\mathsf{Stp}$; (**2**) The key generation algorithm $\mathsf{PBMS}_{\mathsf{RSA}}.\mathsf{KGen}$ run by a signer runs $\mathsf{PBS}_{\mathsf{RSA}}.\mathsf{KG}$ and outputs $(\mathsf{sk}, \mathsf{vk})$ obtained from $\mathsf{PBS}_{\mathsf{RSA}}.\mathsf{KG}$; (**3**) The key aggregation algorithm $\mathsf{PBMS}_{\mathsf{RSA}}.\mathsf{KAgg}$ takes as input a set of verification keys $\{\mathsf{vk}_1, \ldots, \mathsf{vk}_n\}$ and outputs an aggregated key $\mathsf{avk}{:=}\{(N_1, e_1), \ldots, (N_n, e_n)\}, \mathsf{H}_{\mathsf{PSS}}, \mathsf{par}\}$ (**4**) The interactive protocol between the user $\mathsf{PBMS}_{\mathsf{RSA}}.\mathcal{U}$ and the signers $\mathsf{PBMS}_{\mathsf{RSA}}.\mathcal{S}_1, \ldots, \mathsf{PBMS}_{\mathsf{RSA}}.\mathcal{S}_n$ is executed opening n sessions, one with each signer for which it runs multiple times $\mathsf{PBS}_{\mathsf{RSA}}.\mathsf{U}$. For each signer i, the algorithm $\mathsf{PBS}_{\mathsf{RSA}}.\mathsf{UBld}$ takes as input the verification key vk_i from avk along with the same message m and the same predicate description φ. Each signer runs $\mathsf{PBMS}_{\mathsf{RSA}}.\mathcal{S}$ with as input its own secret key and a message received from the user. For each signer i, $\mathsf{PBS}_{\mathsf{RSA}}.\mathsf{UFin}$ outputs a partial signature σ_i. Once all sessions with the signers are terminated, $\mathsf{PBMS}_{\mathsf{RSA}}.\mathcal{U}$ outputs a signature $\sigma{:=}\{\sigma_1, \ldots, \sigma_n\}$. (**5**) The verification algorithm $\mathsf{PBMS}_{\mathsf{RSA}}.\mathsf{Ver}$ takes as input the aggregated key avk, the message m and the signature σ. Then, for every partial signature $\sigma_i \in \{\sigma_1, \ldots, \sigma_n\}$, the algorithm extracts the i-th verification key from avk and runs $\mathsf{PBS}_{\mathsf{RSA}}.\mathsf{Vry}$ passing as input vk_i, m and σ_i. If at least one execution of $\mathsf{PBS}_{\mathsf{RSA}}.\mathsf{Vry}$ outputs 0 then $\mathsf{PBMS}_{\mathsf{RSA}}.\mathsf{Ver}$, otherwise it outputs 1.

Since the construction of PBMS is mainly an iteration of multiple executions of $\mathsf{PBS}_{\mathsf{RSA}}$, the security of the OMUF and the blindness properties are based on the security of the OMUF and blindness property of $\mathsf{PBS}_{\mathsf{RSA}}$, which are shown to be secure, respectively, in Thm. 2 and in Thm. 1.

IV. Experiments

Here we show experimentally that **ACTS** can be used for real-world scenarios. Specifically, we tested a scenario in which a user obtains from a notary a standard RSA-PSS signature of a certificate of attendance released as a PDF file which has been downloaded from a server with TLS (the user receives the PDF in TLS encrypted with AES).

The results of the experiments, presented in Sec. IV-B, prove the viability of our approach, however the resources necessary to generate the proof are quite significant. In Sec. IV-C we discuss a technique to reduce the necessary resources through which users can flexibly generate proofs for downloaded data of virtually any size, with computation time scaling according to the resources available to them.

The most resource-intensive part of our experiment is the generation of the NIZK proof in line 5 of the algorithm $\mathsf{PBS}_{\mathsf{RSA}}.\mathsf{UBld}$ in Fig. 4. This proof attests that the downloaded PDF message, encrypted using AES in counter mode (as in TLS), is the same message that is blinded according to the RSA-BSSA standard defined by the IETF in [9].

Finally, the blind signature produced by the notary is unblinded and embedded into the PDF in a way that allows standard software, in our case Adobe Acrobat Reader, to verify it. All components used to evaluate the performance of the protocol of Fig. 4 are available on GitHub [28].

A. Main Technical Choices

The RSA-BSSA signature protocol defined in [9] is implemented in Rust in a library available on Github [29].

We used Circom to write the NIZK circuit of $\mathsf{PBS}_{\mathsf{RSA}}$[7] and we generated the proof with two different approaches: a Vole-in-the-Head (VitH) proof system [30] and Groth16 proof

[7] We used Circom both to write the circuit and to compile it into a Rank 1 Constraint System (R1CS).

system [31]. For the former, we used a recent Rust library available on Github [32]. For the latter, we used Snarkjs to set up the proof system and Rapidsnark to generate the proof. We chose these two approaches because VitH is a recent proof system that guarantees by design scalability and low memory usage while Groth16 offers succinct proofs, fast verification and it is widely adopted in many application scenarios; this has fostered the development of efficient frameworks (such as Rapidsnark).

In $\mathsf{PBS}_{\mathsf{RSA}}$, during the execution of the algorithm $\mathsf{PBS}_{\mathsf{RSA}}.\mathsf{UBld}$ in Fig. 4, the message is first encoded with $\mathsf{H}_{\mathsf{PSS}}.\mathsf{Enc}$. This encoding works as follows: the message is first hashed with SHA256 (or one of the other SHA-like hash functions) and then the output is padded according to the PSS specification. Then, this padded value is given as input to a mask generation function, and its result is given as output by $\mathsf{H}_{\mathsf{PSS}}.\mathsf{Enc}$. In $\mathsf{PBS}_{\mathsf{RSA}}.\mathsf{UBld}$, this value is finally randomized with a modular exponentiation. We instantiated the circuit of $\mathsf{PBS}_{\mathsf{RSA}}.\mathsf{UBld}$ adopting some Circom implementations already available online, such as [33] for SHA256 and [34] for modular RSA exponentiation. The AES counter mode circuit used in TLS1.2 and TLS1.3, relys on the Circom implementation available online [35]. The embedding of the RSA-PSS standard signature in the PDF has been implemented using Pyhanko [36].

Compliance with standard PDF signatures. A PDF is a structured document format composed of a sequence of objects, organized into a body, a cross-reference table and a trailer. The body contains the actual content (text, images, annotations) as well as metadata and form fields. Digital signatures can be stored as `/Sig` objects, typically referenced within the document `/AcroForm` or `/Annots` entries. The cryptographic signature itself is embedded in the `/Contents` field of the `/Sig` object and formatted as a CMS structure (PKCS #7 SignedData). To ensure a consistent and verifiable signature process, the PDF standard ISO 32000-1/2:2020 [37] defines the `/ByteRange` field, which specifies the exact ranges of bytes in the file that are included in the computation of a digest. The signer computes the cryptographic hash (e.g., using SHA256) over these ranges, embeds it in a structure of signed attributes (including content type and signing time), and then signs this structure. The resulting CMS object (PKCS#7 SignedData), which contains the signature, the attributes and optionally the certificate, is inserted into the `/Contents` field. This procedure guarantees compatibility with existing PDF viewers and signature verification tools, such as Adobe Acrobat Reader. In the Circom circuit, we have implemented this process by first computing the SHA256 hash on the PDF and then computing the SHA256 hash on the SignedData object (which contains the previously computed hash). This second hash value is used as input for the rest of $\mathsf{PBS}_{\mathsf{RSA}}.\mathsf{UBld}$, implemented in the circuit.

B. Experimental results

We executed the complete $\mathsf{PBS}_{\mathsf{RSA}}$ protocol on a realistic template of a PDF file that certifies attendance to a conference (see Fig. 10, Appendix App. B). The size of the document is 26KB and contains both text and images.

The most challenging part of the process is the generation of the NIZK proof. Our circuit operates receiving as input the PDF file, the CMS object (PKCS#7 SignedData), the randomness required to compute the blinding value and PSS encoding, the AES encryption key, an initialization vector for AES, the RSA modulus and the exponent.

The circuit performs the following steps: (1) it computes the SHA256 hash of the PDF; (2) it computes the SHA256 hash of the SignedData object, which includes the hash obtained in step (1); (3) it executes the blinding algorithm $\mathsf{PBS}_{\mathsf{RSA}}.\mathsf{UBld}$; (4) it performs AES encryption in counter mode on the PDF; and (5) it returns as output the ciphertext and the blinded message. The signer then checks that the two output values match those received from the user.

Note that TLS 1.2 and TLS 1.3 work, among other schemes, with AES-GCM. In AES-GCM, every block of plaintext is encrypted with AES-CTR and, then, each block of ciphertext is authenticated in the following way: the ciphertext is multiplied in a Galois field and XORed with a running accumulator. This process combines all ciphertext blocks and additional authenticated data (AAD) into a single authentication tag. This authentication tag is finally XORed with the AES encryption of the first counter of AES-CTR producing the final authentication tag. To lighten the computation of the NIZK, we leave to the notary the verification of the GCM authentication tag because it does not disclose any information on the plaintext; the user only has to provide to the notary a NIZK that the AES encryption of the first counter has been correctly computed, which can be computed with very few computational resources.

To optimize the memory required for the generation of the proof, we used a VitH-based proof system. Performance metrics show that for a real-world PDF file (see Fig. 10) of 26.7 KB, we can generate a valid proof that can be successfully verified by Adobe Acrobat Reader, as shown in Fig. 10 .

In this initial implementation, the compiled circuit size is 34.69 GB and the proof generation needs 328.071 seconds and 201.65 GB of RAM. We emphasize that this is not a fundamental limitation of our approach, and indeed in the next section we discuss some optimization techniques that can strongly reduce memory requirements to largely fit within the 32 GB available on commodity hardware (see Tab. I).

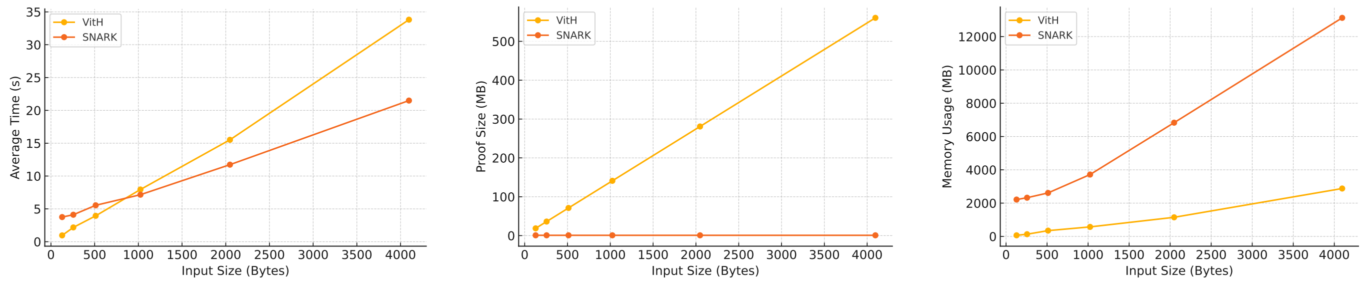
C. Dealing with large data

A valid criticism of our feasibility results concerns the amount of RAM used in the previous experiment, which may not be available to most users. Users usually own devices with much less resources (we consider < 32 GB of RAM). In addition, it is unclear how to run our protocol on larger PDF files, which are quite common in practice, without requiring a very large amount of RAM avaiable to few.

Recently, in [38, 39], the authors demonstrated that it is possible to design efficient proof systems for images, requiring only few GBs of RAM to compute a proof for images up to $\sim$

Input (bytes)	Constraints	Circuit Size (MB)	Approach	Proving Time (s)	RAM (GB)	Proof Size
128	530,793	135.73	VitH	0.96	0.07	18.69 MB
			SNARK	3.75	2.21	800 bytes
256	1,032,665	265.05	VitH	2.18	0.13	36.17 MB
			SNARK	4.11	2.33	800 bytes
512	2,036,409	524.82	VitH	3.94	0.35	71.13 MB
			SNARK	5.54	2.61	800 bytes
1024	4,043,897	1048.96	VitH	7.95	0.57	141.02 MB
			SNARK	7.16	3.72	800 bytes
2048	8,058,873	2115.58	VitH	15.52	1.15	280.82 MB
			SNARK	11.73	6.83	800 bytes
4096	16,088,825	4322.24	VitH	33.81	2.88	560.42 MB
			SNARK	21.49	13.13	800 bytes

TABLE I: Benchmark results comparing proof generation time, memory usage, and proof size for NIZK proofs using the VitH and SNARK backends across varying input sizes.



(a) Proving time with increasing input size. (b) Proof size with increasing input size. (c) Memory usage with increasing input size.

Fig. 5: Benchmark comparison between VitH and SNARK approaches.

30 MP (in raw RGB 30MP corresponds to $\sim$ 90 MB), on the order of minutes. Specifically, the authors of [38] exploit the iterative nature of SHA256. Indeed, SHA256 works in rounds and, in each round, the algorithm hashes a portion of the input message.The authors of [38] devised a proof for each round (or set of rounds) of SHA256 using only the necessary portion of the image as input.

A similar approach can be extended to AES in Counter Mode which also follows an iterative approach, processing a 128 bit block of plaintext at a time. In each round, a counter value is first encrypted using the AES key, producing a keystream block, which is then XORed with the corresponding plaintext block to produce the ciphertext. As in the case of SHA256, it is therefore possible to generate a proof for each encryption round independently by feeding the circuit only with the necessary portion of the message and the corresponding counter input, thus significantly reducing memory usage during proof generation. That is, by directly leveraging and extending to AES the technique of [38], the user can combine these two proofs: a) the correct computation of the SHA256 round on the original file, and b) the correctness of the AES encryption round. Note that, following [38], our approach generalizes to any Merkle-Damgård/Sponge hash function and encryption through block ciphers. We use SHA256 and AES-GCM because they are standardized for RSA-PSS signatures and TLS. We note that if different encryption/hash functions are used (e.g., more SNARK-friendly functions such as Poseidon hash and/or Ciminion encryption), proof generation would be significantly faster.

A natural question is how such a approach impacts on the performance of proof generation. For this reason, we ran a set of experiments aiming to measure the trend of such performance with increasing input size to the circuit. In particular, we implement a circuit that takes as input a message, an AES encryption key and an initialization vector for AES and, then, computes 1) the SHA256 algorithm on the message and 2) the AES in Counter Mode encryption algorithm on the same message. We have generated the NIZK for this circuit for both VitH and Groth16 proof systems and compared their performances. The results are reported in Tab. I and in Fig. 5.

As shown in Fig. 5c, the real advantage of using VitH is that the use of memory increases linearly with the size of the input and uses much less memory with respect to the Groth16 proof generation. This comes at the cost of a larger proof size (as shown in Fig. 5b) that increases linearly with the size of the input, while the proof size of Groth16 remains constant. Finally, Fig. 5a shows that the proof generation time grows linearly with respect to the input size for both proof systems.

In general, with the technique described in this section, our approach became inherently scalable and applicable to a wide range of use-cases: each user can independently generate proofs tailored to their specific requirements and computational resources, enabling the protocol to accommodate a broad class of users, including those with limited hardware. Users can flexibly generate proofs for downloaded data of virtually any size, with computation time scaling according to the resources available to them.

That is, using this approach memory usage remains constant regardless of input size, since each chunk requires the same amount of memory (see Tab. I). Consequently, the practical bound is determined by the available hardware and the user's proof size constraints: more memory permits larger (and therefore fewer) chunks, reducing the number of per-chunk proofs to compute, whereas users with limited hardware can still produce a proof by processing many very small chunks, at the cost of increased proof size.

V. Conclusion

In this work, we presented **ACTS**, a distributed architecture that connects standard TLS servers with the Web3, maintaining untouched the software of existing servers and verifiers.

Our approach, building on top of DECO and of predicate blind signature schemes, makes progress towards closing the gap left open by previous works by enabling verification through standard digital signatures. This removes the need for complex cryptographic tools on the verifier's side and supports immediate integration with existing systems.

By making standard verification possible, **ACTS** can be used in a wide range of real-world domains that could benefit from this technology, potentially leading to widespread adoption. In our view, this represents a major step towards a large-scale adoption of Web3 technologies. In particular, our system enables Web3 applications where smart contracts verify the signatures of messages certified by a notary about data stored in Web2 servers.

We also described how to decentralize the notary service through multisignatures. Improving the usability of this setting is a great challenge for future work. Moreover, given that there currently exists no post-quantum (PQ) secure version of DECO that can be applied to PQ-secure TLS, and practical blind signatures for standardized PQ schemes (e.g., Falcon, Dilithium, SPHINCS+) do not yet exist, we opted for RSA-PSS signatures and (non-PQ) TLS-DECO. Nevertheless, investigating PQ PBS alternatives for standardized schemes that can be used with a future PQ-secure version of DECO is an interesting direction for future work.

Acknowledgments

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU. All authors are members of the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM).

References

[1] F. Zhang, D. Maram, H. Malvai, S. Goldfeder, and A. Juels, "DECO: Liberating web data using decentralized oracles for TLS," in *ACM CCS 2020*, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds. ACM Press, Nov. 2020, pp. 1919–1938.

[2] X. Xie, K. Yang, X. Wang, and Y. Yu, "Lightweight authentication of web data via garble-then-prove," in *USENIX Security 2024*, D. Balzarotti and W. Xu, Eds. USENIX Association, Aug. 2024. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/xie-xiang

[3] J. Ernstberger, J. Lauinger, Y. Wu, A. Gervais, and S. Steinhorst, "ORIGO: Proving provenance of sensitive data with constant communication," Cryptology ePrint Archive, Report 2024/447, 2024, accepted at Proceedings on Privacy Enhancing Technologies Symposium 2025. [Online]. Available: https://eprint.iacr.org/2024/447

[4] J. Lauinger, J. Ernstberger, A. Finkenzeller, and S. Steinhorst, "Janus: Fast privacy-preserving data provenance for TLS 1.3," Cryptology ePrint Archive, Report 2023/1377, 2023, accepted at Proceedings on Privacy Enhancing Technologies Symposium 2025. [Online]. Available: https://eprint.iacr.org/2023/1377

[5] D. Chaum, "Blind signatures for untraceable payments," in *CRYPTO'82*, D. Chaum, R. L. Rivest, and A. T. Sherman, Eds. Plenum Press, New York, USA, 1982, pp. 199–203.

[6] ——, "Elections with unconditionally-secret ballots and disruption equivalent to breaking RSA," in *EUROCRYPT'88*, ser. LNCS, C. G. Günther, Ed., vol. 330. Springer, Berlin, Heidelberg, May 1988, pp. 177–182.

[7] S. Brands, "Untraceable off-line cash in wallets with observers (extended abstract)," in *CRYPTO'93*, ser. LNCS, D. R. Stinson, Ed., vol. 773. Springer, Berlin, Heidelberg, Aug. 1994, pp. 302–318.

[8] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "TumbleBit: An untrusted bitcoin-compatible anonymous payment hub," in *NDSS 2017*. The Internet Society, Feb. / Mar. 2017.

[9] F. Denis, F. Jacobs, and C. A. Wood, "RSA Blind Signatures," RFC 9474, Oct. 2023. [Online]. Available: https://www.rfc-editor.org/info/rfc9474

[10] M. Bellare and P. Rogaway, "The exact security of digital signatures: How to sign with RSA and Rabin," in *EUROCRYPT'96*, ser. LNCS, U. M. Maurer, Ed., vol. 1070. Springer, Berlin, Heidelberg, May 1996, pp. 399–416.

[11] ——, "Pss: Provably secure encoding method for digital signatures," 1998, submission to IEEE P1363, August 1998. [Online]. Available: https://www.cs.ucdavis.edu/~rogaway/papers/exact.html

[12] G. Fuchsbauer and M. Wolf, "Concurrently secure blind Schnorr signatures," in *EUROCRYPT 2024, Part II*, ser. LNCS, M. Joye and G. Leander, Eds., vol. 14652. Springer, Cham, May 2024, pp. 124–160.

[13] I. Karantaidou, O. Renawi, F. Baldimtsi, N. Kamarinakis, J. Katz, and J. Loss, "Blind multisignatures for anonymous tokens with decentralized issuance," in *ACM CCS 2024*, B. Luo, X. Liao, J. Xu, E. Kirda, and D. Lie, Eds. ACM Press, Oct. 2024, pp. 1508–1522.

[14] E. C. Crites, C. Komlo, M. Maller, S. Tessaro, and C. Zhu, "Snowblind: A threshold blind signature in pairing-free groups," in *CRYPTO 2023, Part I*, ser. LNCS, H. Handschuh and A. Lysyanskaya, Eds., vol. 14081. Springer, Cham, Aug. 2023, pp. 710–742.

[15] A. Lehmann, P. Nazarian, and C. Özbay, "Stronger security for threshold blind signatures," in *EUROCRYPT 2025, Part II*, ser. LNCS, S. Fehr and P.-A. Fouque, Eds., vol. 15602. Springer, Cham, May 2025, pp. 335–364.

[16] K. Yang, P. Sarkar, C. Weng, and X. Wang, "QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field," in *ACM CCS 2021*, G. Vigna and E. Shi, Eds. ACM Press, Nov. 2021, pp. 2986–3001.

[17] S. Celi, A. Davidson, H. Haddadi, G. Pestana, and J. Rowell, "DiStefano: Decentralized infrastructure for sharing trusted encrypted facts and nothing more," in *NDSS 2025*. The Internet Society, Feb. 2025.

[18] K. Y. Chan, H. Cui, and T. H. Yuen, "Dido: Data provenance from restricted tls 1.3 websites," in *Information Security Practice and Experience: 18th International Conference, ISPEC 2023*. Berlin, Heidelberg: Springer-Verlag, 2023, p. 154–169.

[19] Z. Luo, Y. Jia, Y. Shen, and A. Kate, "Proxying Is Enough: Security of Proxying in TLS Oracles and AEAD Context Unforgeability," in *7th Conference on Advances in Financial Technologies (AFT 2025)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 354.

Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025, pp. 4:1–4:24.

[20] M. Abe and E. Fujisaki, "How to date blind signatures," in *ASIACRYPT'96*, ser. LNCS, K. Kim and T. Matsumoto, Eds., vol. 1163. Springer, Berlin, Heidelberg, Nov. 1996, pp. 244–251.

[21] S. Tessaro and C. Zhu, "Short pairing-free blind signatures with exponential security," in *EUROCRYPT 2022, Part II*, ser. LNCS, O. Dunkelman and S. Dziembowski, Eds., vol. 13276. Springer, Cham, May / Jun. 2022, pp. 782–811.

[22] M. Fischlin, "Round-optimal composable blind signatures in the common reference string model," in *CRYPTO 2006*, ser. LNCS, C. Dwork, Ed., vol. 4117. Springer, Berlin, Heidelberg, Aug. 2006, pp. 60–77.

[23] A. Lysyanskaya, "Security analysis of RSA-BSSA," in *PKC 2023, Part I*, ser. LNCS, A. Boldyreva and V. Kolesnikov, Eds., vol. 13940. Springer, Cham, May 2023, pp. 251–280.

[24] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, 3rd ed. Chapman and Hall, CRC Press, 2014.

[25] M. Fischlin, "Communication-efficient non-interactive proofs of knowledge with online extractors," in *CRYPTO 2005*, ser. LNCS, V. Shoup, Ed., vol. 3621. Springer, Berlin, Heidelberg, Aug. 2005, pp. 152–168.

[26] R. Canetti, "Universally composable security," *J. ACM*, vol. 67, no. 5, Sep. 2020. [Online]. Available: https://doi.org/10.1145/3402457

[27] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in *USENIX Security 2014*, K. Fu and J. Jung, Eds. USENIX Association, Aug. 2014, pp. 781–796. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson

[28] "Github project," https://github.com/marcozecchini/blindRSANotary, 2025.

[29] F. Denis, "Blind rsa signatures in pure rust," https://github.com/jedisct1/rust-blind-rsa-signatures, 2021.

[30] C. Baum, L. Braun, C. Delpech de Saint Guilhem, M. Klooß, E. Orsini, L. Roy, and P. Scholl, "Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head," in *CRYPTO 2023, Part V*, ser. LNCS, H. Handschuh and A. Lysyanskaya, Eds., vol. 14085. Springer, Cham, Aug. 2023, pp. 581–615.

[31] J. Groth, "On the size of pairing-based non-interactive arguments," in *EUROCRYPT 2016, Part II*, ser. LNCS, M. Fischlin and J.-S. Coron, Eds., vol. 9666. Springer, Berlin, Heidelberg, May 2016, pp. 305–326.

[32] Holonym Foundation, "Vole in the head aimed for compatibility with popular dsls," https://github.com/holonym-foundation/vole-zk-prover, 2024.

[33] iden3, "Circomlib: Sha256 circuit implementation," https://github.com/iden3/circomlib/blob/master/circuits/sha256/, 2020.

[34] "Rsa signature verification circuit in circom," https://github.com/zkp-application/circom-rsa-verify, 2023.

[35] Electron Labs, "Aes circuit in circom," https://github.com/Electron-Labs/aes-circom, 2023.

[36] M. Valvekens, "pyhanko: Toolkit for creating and verifying pdf digital signatures," https://github.com/MatthiasValvekens/pyHanko, 2020.

[37] International Organization for Standardization, "Portable Document Format: Part 2," https://www.iso.org/standard/75839.html, 2020.

[38] P. Della Monica, I. Visconti, A. Vitaletti, and M. Zecchini, "Trust Nobody: Privacy-Preserving Proofs for Edited Photos with Your Laptop," in *2025 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2025, pp. 14–14.

[39] S. Dziembowski, S. Ebrahimi, and P. Hassanizadeh, "VIMz: Verifiable image manipulation using folding-based zkSNARKs," Cryptology ePrint Archive, Paper 2024/1063, 2024, accepted at Privacy Enhancing Technologies Symposium (PETS) 2025. [Online]. Available: https://eprint.iacr.org/2024/1063

[40] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko, "The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme," *Journal of Cryptology*, vol. 16, no. 3, pp. 185–215, Jun. 2003.

[41] K. Moriarty, B. Kaliski, J. Jonsson, and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2," RFC 8017, Nov. 2016. [Online]. Available: https://www.rfc-editor.org/info/rfc8017

[42] B. Bünz, A. Chiesa, P. Mishra, and N. Spooner, "Recursive proof composition from accumulation schemes," in *TCC 2020, Part II*, ser. LNCS, R. Pass and K. Pietrzak, Eds., vol. 12551. Springer, Cham, Nov. 2020, pp. 1–18.

[43] A. Chiesa, D. Ojha, and N. Spooner, "Fractal: Post-quantum and transparent recursive proofs from holography," in *EUROCRYPT 2020, Part I*, ser. LNCS, A. Canteaut and Y. Ishai, Eds., vol. 12105. Springer, Cham, May 2020, pp. 769–793.

[44] S. Goldberg, L. Reyzin, O. Sagga, and F. Baldimtsi, "Efficient noninteractive certification of RSA moduli and beyond," in *ASIACRYPT 2019, Part III*, ser. LNCS, S. D. Galbraith and S. Moriai, Eds., vol. 11923. Springer, Cham, Dec. 2019, pp. 700–727.

Appendix A
Standard Notation and Additional Preliminaries

Notation. Given $n \in \mathbb{N}^+$ we write $[n]$ for $\{1, \ldots, n\}$. When x is chosen randomly in $\mathcal{X}$, we write $x \leftarrow\!\!\$\ \mathcal{X}$. When $\mathcal{A}$ is an algorithm, we write $y \leftarrow \mathcal{A}(x)$ to denote a run of $\mathcal{A}$ on input x and output y; if $\mathcal{A}$ is randomized, then y is a random variable and $\mathcal{A}(x; \rho)$ denotes a run of $\mathcal{A}$ on input x and random coins $\rho \in \{0, 1\}^*$.

An empty list is initialized via $\mathbf{a}{:=}[\,]$. A value x is appended to a list $\mathbf{a}$ via $\mathbf{a} = \mathbf{a} \leftarrow \mathbf{a} \| x$. The size of $\mathbf{a}$, representing the number of elements in the list, is denoted by $|\mathbf{a}|$. We denote the i-th element of $\mathbf{a}$ by $\mathbf{a}_i$. Given the list $\mathbf{a}$, attempts to access a position $i \notin [|\mathbf{a}|]$ return the empty symbol ε. A tuple of elements is denoted as $x{:=}(a, \ldots, z)$ (where $a, \ldots, z$ are the elements of the tuple) and $x[i]$ denotes the i-th element, which we set to ε if it does not exist.

We denote the security parameter by $\lambda \in \mathbb{N}$. A function $\mathsf{negl}(\lambda)$ is negligible in λ (or shortly negligible) if it vanishes faster than the inverse of any polynomial (i.e., for any constant $c > 0$ for sufficiently large λ it holds that $\mathsf{negl}(\lambda) \leq \lambda^{-c}$). A machine is said to be probabilistic polynomial time (PPT), or efficient, if it is randomized and its number of steps is polynomial in the input size.

For a random variable $\mathbf{X}$, we write $\Pr[\mathbf{X} = x]$ for the probability that $\mathbf{X}$ takes a particular value x. A distribution ensemble $\mathbf{X} = \{\mathbf{X}(\lambda)\}_{\lambda \in \mathbb{N}}$ is an infinite sequence of random variables indexed by the security parameter $\lambda \in \mathbb{N}$. Two distribution ensembles $\mathbf{X} = \{\mathbf{X}(\lambda)\}_{\lambda \in \mathbb{N}}$ and $\mathbf{Y} = \{\mathbf{Y}(\lambda)\}_{\lambda \in \mathbb{N}}$ are said to be *computationally indistinguishable*, denoted by $\mathbf{X} \overset{c}{\approx} \mathbf{Y}$, if for every non-uniform PPT algorithm D there exists a negligible function $\mathsf{negl}(\lambda)$ such that:

$$|\Pr[\mathsf{D}(\mathbf{X}(\lambda)) = 1] - \Pr[\mathsf{D}(\mathbf{Y}(\lambda)) = 1]| \leq \mathsf{negl}(\lambda)$$

To enhance the readability of pseudocode, if a value a "implicitly defines" values $(b_1, b_2, \ldots)$ (that is, these can be parsed or anyway trivially obtained from a), we write $(b_1, b_2, \ldots) :\subseteq a$.

A. The RSA Hardness Assumption

Definition 4. *An* RSA generation algorithm RSAGen *is a* PPT *algorithm that takes as input a security parameter λ in unary and returns (N, e, d), where N is a product of two distinct $(\lambda/2)$-bit primes p and q, $e \in \mathbb{Z}^*_{\phi(N)}$ is a public exponent, and d is the corresponding secret exponent such that $ed \equiv 1 \pmod{\phi(N)}$.*

Definition 5 ([40, Definition 2.1])**.** *Consider the game* RSA*, described in Fig. 6. An* RSA generation algorithm RSAGen *is considered (to output RSA instances that are) hard if for every* PPT *adversary $\mathcal{A}$ the advantage $\mathsf{Adv}^{\mathrm{rsa}}_{\mathsf{RSAGen}}(\mathcal{A}, \lambda)$ is negligible in λ where:*

$$\mathsf{Adv}^{\mathrm{rsa}}_{\mathsf{RSAGen}}(\mathcal{A},\lambda) := \Pr\left[\mathrm{RSA}^{\mathcal{A}}_{\mathsf{RSAGen}}(\lambda) = 1\right]$$

```
Game $\mathrm{RSA}^{\mathcal{A}}_{\mathsf{RSAGen}}(\lambda)$
1 : $(N, e, d) \leftarrow \mathsf{RSAGen}(1^\lambda); y \leftarrow\!\!\$\ \mathbb{Z}_N$
2 : $x \leftarrow \mathcal{A}(N, e, y)$
3 : return $(x^e = y)$
```

Fig. 6: The RSA game.

B. Signature Schemes

A signature scheme is a tuple of efficient algorithms $\mathsf{Sig} = (\mathsf{Sig.KG}, \mathsf{Sig.Sig}, \mathsf{Sig.Vry})$ where:

- $\mathsf{Sig.KG}(1^\lambda) \rightarrow (\mathsf{sk}, \mathsf{vk})$ on input the security parameter, outputs a signing key sk and a verification key vk.
- $\mathsf{Sig.Sig}(\mathsf{sk}, m) \rightarrow \sigma$ on input a signing key sk and a message $m \in \mathcal{M}_{\mathsf{Sig}}$, outputs a signature σ.
- $\mathsf{Sig.Vry}(\mathsf{vk}, m, \sigma) \rightarrow 0/1$ is deterministic and on input a verification key vk, a message m and a signature σ, outputs 1 if σ is valid and 0 otherwise.

and such that Correctness and Unforgeability as defined below hold.

Definition 6 (Correctness)**.** *A signature scheme* Sig *is perfectly correct if for all* $\lambda \in \mathbb{N}$ *and* $m \in \mathcal{M}_{\mathsf{Sig}}$*:*

$$\Pr\left[1 = \mathsf{Sig.Vry}(\mathsf{vk}, m, \sigma) \;\middle|\; \begin{array}{c}(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Sig.KG}(1^\lambda),\\ \sigma \leftarrow \mathsf{Sig.Sig}(\mathsf{sk}, m),\end{array}\right] = 1$$

Definition 7 (Unforgeability)**.** *Consider the game* EUF-CMA*, described in Fig. 7. A signature scheme* Sig *satisfies existential unforgeability under chosen-message attacks (*SUF-CMA*) if for every* PPT *adversary* $\mathcal{A}$ *the advantage* $\mathsf{Adv}^{\text{euf-cma}}_{\mathsf{Sig}}(\mathcal{A},\lambda)$ *is negligible in* λ *where* $\mathsf{Adv}^{\text{euf-cma}}_{\mathsf{Sig}}(\mathcal{A},\lambda) := \Pr\left[\text{EUF-CMA}^{\mathcal{A}}_{\mathsf{Sig}}(\lambda) = 1\right]$.

1) RSA Signature: We focus here on the RSA-PSS variant [10, 11], which is nowadays considered the standard RSA signature scheme in practice. In RSA-PSS, the signature on a message m is computed as the RSA inverse of a special encoding of m, called the PSS encoding, denoted by m'.

The PSS encoding employs two cryptographic hash functions, $\mathcal{H}$ and $\mathcal{G}_{\mathsf{MGF}}$, both modeled as random oracles in the security analysis of RSA-PSS. Following [10], although these functions have similar input-output specifications and security requirements, it is helpful to instantiate them separately in practice. This is because $\mathcal{H}$ typically maps a potentially long input to a fixed-length output, whereas $\mathcal{G}_{\mathsf{MGF}}$ (mask generation function) expands a short seed into a longer output.

We refer the reader to the IETF standard [41] or to [23, Section 3] for a detailed explanation of the PSS encoding, which is nevertheless conceptually simple. In our treatment, we abstract this encoding process as a black-box function that maps a message in $\{0,1\}^*$ (along with a random salt in $\{0,1\}^\lambda$) to an encoded message in $\mathbb{Z}_N^*$ (and vice-versa), suitable for RSA signatures.

Definition 8. *A (target-range)* PSS *function generator* PSSGen *is a* PPT *algorithm that, given a modulus* $N \in \mathbb{N}^+$*, outputs the description of a pair of deterministic algorithms* $\mathsf{H}_{\mathsf{PSS}} := (\mathsf{H}_{\mathsf{PSS}}.\mathsf{Enc}, \mathsf{H}_{\mathsf{PSS}}.\mathsf{Dec})$*, where* $\mathsf{H}_{\mathsf{PSS}}.\mathsf{Enc} : \{0,1\}^* \times \{0,1\}^\lambda \rightarrow \mathbb{Z}_N^*$ *and* $\mathsf{H}_{\mathsf{PSS}}.\mathsf{Dec} : \mathbb{Z}_N^* \rightarrow \{0,1\}^*$*. Both algorithms internally make black-box use of the (sampled) functions* $\mathcal{H}$ *and* $\mathcal{G}_{\mathsf{MGF}}$.

Following the recent work of Fuchsbauer and Wolf [12], which argues for the uncontroversial assumption that the Schnorr signature scheme remains secure when instantiated with concrete hash functions (e.g., SHA-2 family) instead of ideal random oracles, we adopt a similar approach for RSA-PSS. Their justification is understandably based on the widespread use of Schnorr signatures in practice. Given the even more extensive deployment of RSA-PSS over the years, we make the following assumption in our security analysis[8].

Assumption 1. *There exists an RSA key generation algorithm* RSAGen *and a PSS function generator* PSSGen *such that the RSA-PSS signature scheme in Fig. 8 is unforgeable according to Def. 7. In particular, for every* PPT *adversary* $\mathcal{A}$*, the advantage* $\mathsf{Adv}^{\text{euf-cma}}_{\mathsf{RSA}[\mathsf{RSAGen},\mathsf{PSSGen}]}(\mathcal{A},\lambda)$ *is negligible in* λ.

C. Non-Interactive Zero-Knowledge Schemes

We define a non-interactive zero-knowledge (NIZK) argument/proof of knowledge system, with respect to a *parameterized relation* $\mathsf{R} : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}$ which is a binary relation. Given a statement $\mathbb{x}$ we call $\mathbb{w}$ a witness if $\mathsf{R}(\mathbb{x}, \mathbb{w}) = 1$, and define the language $\mathcal{L}_{\mathsf{R}} := \{\mathbb{x} \mid \exists \mathbb{w} : \mathsf{R}(\mathbb{x}, \mathbb{w}) = 1\}$. A NIZK argument of knowledge system for a relation R is a tuple of efficient PPT algorithms $\mathsf{PS}[\mathsf{R}] = (\mathsf{PS.Stp}, \mathsf{PS.Prv}, \mathsf{PS.Vry}, \mathsf{PS.Sim}, \mathsf{PS.Ext})$ where:

- $\mathsf{PS.Stp}(1^\lambda) \rightarrow (\mathsf{crs}, \tau)$ on input the security parameter, returns a common reference string (CRS) crs and a trapdoor τ[9].
- $\mathsf{PS.Prv}(\mathsf{crs}, \mathbb{x}, \mathbb{w}) \rightarrow \pi$ on input a CRS crs, a statement $\mathbb{x}$ and a witness $\mathbb{w}$, outputs a proof π.
- $\mathsf{PS.Vry}(\mathsf{crs}, \mathbb{x}, \pi) \rightarrow 0/1$ is deterministic and on input a CRS crs, a statement $\mathbb{x}$ and a proof π, outputs 1 (accept) or 0 (reject).
- $\mathsf{PS.Sim}(\mathsf{crs}, \tau, \mathbb{x}) \rightarrow \pi$ on input a CRS crs, a trapdoor τ and a statement $\mathbb{x}$, outputs a proof π.
- $\mathsf{PS.Ext}(\mathsf{crs}, \tau, \mathbb{x}, \pi) \rightarrow \mathbb{w}$ on input a CRS crs, a trapdoor τ, a statement $\mathbb{x}$ and a proof π outputs a string $\mathbb{w} \in \{0,1\}^*$.

[8]Note that making a heuristic assumption that a concrete hash function (e.g., SHA-2) can be safely used as a valid substitute for a random oracle is common in both practical and theoretical cryptographic literature. Moreover, also proving claims about the output of such hash functions is standard both in practice (e.g., in ZK-rollup protocols) and in theoretical works on recursive SNARKs [42, 43].

[9]Formally, we can view such a trapdoor as a tuple consisting of a simulation trapdoor and an extraction trapdoor, which are useful to the simulator for the Zero-Knowledge property and to the extractor for the Proof of Knowledge property. To avoid burdening the notation, we assume that the first step of both algorithms is to get the right component of the trapdoor τ needed for their execution.

```
Game $\mathrm{EUF\text{-}CMA}^{\mathcal{A}}_{\mathsf{Sig}}(\lambda)$
  1: $(\mathsf{sk},\mathsf{vk}) \leftarrow \mathsf{Sig.KG}(1^\lambda)$; $\mathcal{Q}{:=}\varnothing$
  2: $(m^*,\sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}}(\mathsf{vk})$
  3: $\mathbf{return}\ ((m^*,\sigma^*) \notin \mathcal{Q} \wedge \mathsf{Sig.Vry}(\mathsf{vk},m^*,\sigma^*) = 1)$

Oracle $\mathsf{Sign}(m)$
  1: $\sigma \leftarrow \mathsf{Sig.Sig}(\mathsf{sk},m)$
  2: $\mathcal{Q} = \mathcal{Q} \cup \{(m,\sigma)\}$
  3: $\mathbf{return}\ \sigma$
```

Fig. 7: The EUF-CMA game.

```
Alg. $\mathsf{RSA.KG}(1^\lambda)$
  1: $(N,e,d) \leftarrow \mathsf{RSAGen}(1^\lambda)$
  2: $\mathsf{H_{PSS}} \leftarrow \mathsf{PSSGen}(N)$
  3: $\mathsf{sk}{:=}(N,e,d,\mathsf{H_{PSS}})$; $\mathsf{vk}{:=}(N,e,\mathsf{H_{PSS}})$
  4: $\mathbf{return}\ (\mathsf{sk},\mathsf{vk})$

Alg. $\mathsf{RSA.Sig}(\mathsf{sk},m)$
  1: $(N,d,\mathsf{H_{PSS}}) :\subseteq \mathsf{sk}$; $r \leftarrow\!\!\$ \{0,1\}^\lambda$
  2: $\mu \leftarrow \mathsf{H_{PSS}.Enc}(m,r)$
  3: $\sigma{:=}\mu^d \pmod N$
  4: $\mathbf{return}\ \sigma$

Alg. $\mathsf{RSA.Vry}(\mathsf{vk},m,\sigma)$
  1: $(N,e,\mathsf{H_{PSS}}) :\subseteq \mathsf{vk}$
  2: $\mu'{:=}\sigma^e \pmod N$
  3: $m' \leftarrow \mathsf{H_{PSS}.Dec}(\mu')$
  4: $\mathbf{return}\ (m' = m)$
```

Fig. 8: The RSA-PSS signature scheme $\mathsf{RSA}[\mathsf{RSAGen},\mathsf{PSSGen}]$ based on a key-pair generator RSAGen and a PSS encoding generator PSSGen.

```
Game $\mathrm{POK}^{\mathcal{A}}_{\mathsf{PS[R]}}(\lambda)$
  1: $(\mathsf{crs},\tau) \leftarrow \mathsf{PS.Stp}(1^\lambda)$
  2: $(\mathbb{x},\pi) \leftarrow \mathcal{A}(\mathsf{crs})$; $\mathbb{w} \leftarrow \mathsf{PS.Ext}(\mathsf{crs},\tau,\mathbb{x},\pi)$
  3: $\mathbf{return}\ (1 = \mathsf{PS.Vry}(\mathsf{crs},\mathbb{x},\pi) \wedge 1 \neq \mathsf{R}(\mathbb{x},\mathbb{w})))$
```

(a) The POK game.

```
Game $\mathrm{ZK}^{\mathcal{A},b}_{\mathsf{PS[R]}}(\lambda)$
  1: $(\mathsf{crs},\tau) \leftarrow \mathsf{PS.Stp}(1^\lambda)$
  2: $b' \leftarrow \mathcal{A}^{\mathsf{Prove}}(\mathsf{crs})$
  3: $\mathbf{return}\ (b = b')$

Oracle $\mathsf{Prove}(\mathbb{x},\mathbb{w})$
  1: $\mathbf{if}\ \mathsf{R}(\mathbb{x},\mathbb{w}) \neq 1$
  2:     $\mathbf{return}\ \bot$
  3: $\pi_0 \leftarrow \mathsf{PS.Prv}(\mathsf{crs},\mathbb{x},\mathbb{w})$
  4: $\pi_1 \leftarrow \mathsf{PS.Sim}(\mathsf{crs},\tau,\mathbb{x})$
  5: $\mathbf{return}\ \pi_b$
```

(b) The ZK game.

Fig. 9: The POK and ZK games, where $\mathcal{L}_{\mathsf{R}}{:=}\{\mathbb{x} \mid \exists \mathbb{w} : \mathsf{R}(\mathbb{x},\mathbb{w}) = 1\}$.

and such that Correctness, Zero-Knowledge and Proof of Knowledge as defined below hold.

Definition 9 (Correctness). *A NIZK argument of knowledge system* $\mathsf{PS[R]}$ *is perfectly correct if for every tuple* $(\mathbb{x},\mathbb{w})$*, such that* $\mathsf{R}(\mathbb{x},\mathbb{w}) = 1$ *and* $\lambda \in \mathbb{N}$*:*

$$\Pr\left[1 = \mathsf{PS.Vry}(\mathsf{crs},\mathbb{x},\pi) \;\middle|\; \begin{array}{c}(\mathsf{crs},\tau) \leftarrow \mathsf{PS.Stp}(1^\lambda),\\ \pi \leftarrow \mathsf{PS.Prv}(\mathsf{crs},\mathbb{x},\mathbb{w})\end{array}\right] = 1$$

Definition 10 (Zero-Knowledge). *Consider the game* ZK*, described in Fig. 9b. A NIZK argument of knowledge system* $\mathsf{PS[R]}$ *is (computationally) zero-knowledge if for every* PPT *adversary* $\mathcal{A}$ *the advantage* $\mathsf{Adv}^{\mathrm{zk}}_{\mathsf{PS[R]}}(\mathcal{A},\lambda)$ *is negligible in* λ *where:*

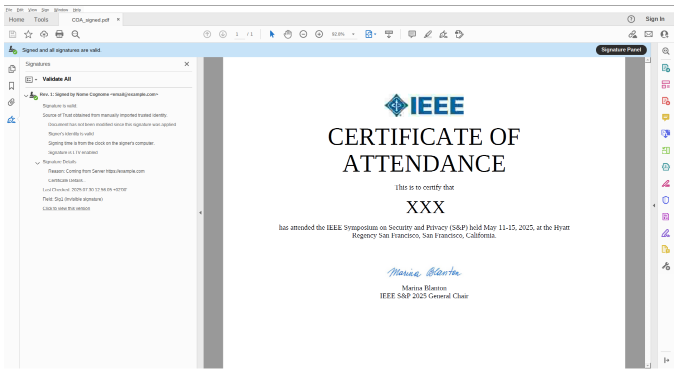
$$\mathsf{Adv}^{\mathrm{zk}}_{\mathsf{PS[R]}}(\mathcal{A},\lambda){:=} \Big|\Pr\Big[\mathrm{ZK}^{\mathcal{A},0}_{\mathsf{PS[R]}}(\lambda) = 1\Big] - \Pr\Big[\mathrm{ZK}^{\mathcal{A},1}_{\mathsf{PS[R]}}(\lambda) = 1\Big]\Big|$$

Definition 11 (Proof of Knowledge). *Consider the game* POK*, described in Fig. 9a. A NIZK argument of knowledge system* $\mathsf{PS[R]}$ *satisfies the (computational) proof of knowledge property if for every* PPT *adversary* $\mathcal{A}$ *the advantage* $\mathsf{Adv}^{\mathrm{pok}}_{\mathsf{PS[R]}}(\mathcal{A},\lambda)$ *is negligible in* λ *where:* $\mathsf{Adv}^{\mathrm{pok}}_{\mathsf{PS[R]}}(\mathcal{A},\lambda){:=}\Pr\Big[\mathrm{POK}^{\mathcal{A}}_{\mathsf{PS[R]}}(\lambda) = 1\Big]$.

When the definition is for every PPT *adversary then the property is also called as argument of knowledge, while in case of non-efficient adversaries it is called proof of knowledge.*

Appendix B
Example of a Realistic Document

Fig. 10 provides a screenshot of Adobe Acrobat Reader successfully verifying the digital signature embedded in a standard PDF. This serves as a concrete confirmation that our protocol produces signatures on real-world documents and that such signatures are fully compatible with standard PDF viewers and adhere to the ISO 32000-1/2:2020 specification.



Signed and all signatures are valid.

Rev. 1: Signed by Nome Cognome <email@example.com>

Signature is valid:

Source of Trust obtained from manually imported trusted identity.

Document has not been modified since this signature was applied

Signer's identity is valid

Signing time is from the clock on the signer's computer.

Signature is LTV enabled

Signature Details

Reason: Coming from Server https://example.com

Certificate Details...

Last Checked: 2025.07.30 12:56:05 +02'00'

Field: Sig1 (invisible signature)

Click to view this version

IEEE

CERTIFICATE OF ATTENDANCE

This is to certify that

XXX

has attended the IEEE Symposium on Security and Privacy (S&P) held May 11-15, 2025, at the Hyatt Regency San Francisco, San Francisco, California.

Marina Blanton

Marina Blanton
IEEE S&P 2025 General Chair

Fig. 10: Screenshot of Adobe Acrobat Reader successfully verifying the digital signature embedded in the PDF, demonstrating compliance with the ISO 32000-1/2:2020 standard.

Appendix C
Proofs of Thms. 1 and 2

Proof of Thm. 1. We give a formal proof that our predicate blind signature scheme $\mathsf{PBS}_{\mathsf{RSA}}$ from Fig. 4 satisfies blindness in Def. 3. We proceed by a sequence of games.

GAME_0. This is game BLD from Fig. 3 with PBS instantiated with $\mathsf{PBS}_{\mathsf{RSA}}$ from Fig. 4, that is, the algorithms PBS.Stp, PBS.UBld and PBS.UFin are replaced by the instantiations defined in Fig. 4. The variables st_0 and st_1 in BLD are set to κ_0 and κ_1 according to the sampling made in $\mathsf{PBS}_{\mathsf{RSA}}$.UBld. We also call it the real game.

GAME_1. In this game, we make the following change: On oracle call Blind, instead of creating a proof via PS.Prv we use the simulator PS.Sim to simulate a proof for the statement $\mathbb{x}$. We show that this change is not efficiently noticeable by defining adversaries $\mathcal{Z}_0$ and $\mathcal{Z}_1$ in that play in game ZK against the $\mathsf{PS}[\mathsf{R}_{\mathsf{Blind}}]$.

We describe the strategy of the adversary $\mathcal{Z}_b$ for $b \in \{0, 1\}$. $\mathcal{Z}_b$ on input crs generated by PS.Stp, simulates the game BLD^b for $\mathcal{A}$, using its oracle Prove (according to Fig. 9b) to obtain the proof π required to answer $\mathcal{A}$'s queries to Blind, the rest of the answer is made by $\mathcal{Z}_b$ as in the real game.

Finally, when $\mathcal{A}_2$ (we recall that $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ in BLD game) outputs its bit b', $\mathcal{Z}_b$ outputs the same bit to its challenger. By the definition of $\mathcal{Z}_b$ for $b \in \{0, 1\}$, we have that when b is equal to 0 in the ZK game then the adversary $\mathcal{Z}_b$ perfectly simulates the real game GAME_0 while when b is equal to 1 in the ZK game then $\mathcal{Z}_b$ perfectly simulates GAME_1. Thus, we have that:

$$\mathsf{Adv}^{\mathrm{zk}}_{\mathsf{PS}[\mathsf{R}_{\mathsf{Blind}}]}(\mathcal{Z}_b, \lambda) := \left|\Pr\left[\text{BLD}^{\mathcal{A},b}_{\mathsf{PBS}_{\mathsf{RSA}}}\right] - \Pr\left[\text{GAME}^{\mathcal{A},b}_1\right]\right|$$

By recalling the definition of $\mathsf{Adv}^{\mathrm{bld}}_{\mathsf{PBS}_{\mathsf{RSA}}}(\mathcal{A}, \lambda)$ and applying the triangular inequality, we have that the advantage of the adversary $\mathcal{A}$ in the BLD is at most equal to:

$$\begin{aligned}&\mathsf{Adv}^{\mathrm{zk}}_{\mathsf{PS}[\mathsf{R}_{\mathsf{Blind}}]}(\mathcal{Z}_0, \lambda) + \mathsf{Adv}^{\mathrm{zk}}_{\mathsf{PS}[\mathsf{R}_{\mathsf{Blind}}]}(\mathcal{Z}_1, \lambda)\\ &\quad + \left|\Pr\left[\text{GAME}^{\mathcal{A},1}_1\right] - \Pr\left[\text{GAME}^{\mathcal{A},0}_1\right]\right|\end{aligned} \tag{2}$$

Reducing GAME_1 to perfect blindness of "plain" blind RSA. For this last step, we follow the same approach as in [23] to prove that the plain version (i.e., without predicates) of blind RSA satisfies the blindness property.

We analyze two cases based on key generation. When the public key is honestly generated by the adversary, as observed in in [5, 40], the use of a blinding factor $\kappa \in [N]$ ensures that the pair (m, σ) is statistically independent of the value t that is observed by the signer. Since we have already established that the proof π provides no additional information, this statistical independence implies blindness.

As noted in [23], the adversary might also generate the public key maliciously (recall that in the BLD game in Fig. 3, these values can be chosen by the adversary) to potentially extract information about low-entropy input messages. Lysyanskaya shows that for any maliciously chosen RSA modulus N and public exponent e, security depends on the PSSGen function. She outlines two approaches for protection: (1) requiring proofs that the signer's public key is honestly generated, in [44], the authors show how to achieve HVZK proofs for statements about the public key, or (2) ensuring the input message contains high-entropy components unknown to the signer before key generation. Our construction follows the latter approach, similar to RSA-BSSA blindness security proof of [23], by providing randomness in the $\mathsf{H}_{\mathsf{PSS}}$ encoding function. This ensures that even with maliciously generated keys, it is not possible to recover the message, and thus not possible to extract any information about the bit b from the input t, just as in the case of honestly generated public keys. We thus have that $\Pr\left[\text{GAME}^{\mathcal{A},1}_1(\lambda) = 1\right] - \Pr\left[\text{GAME}^{\mathcal{A},0}_1(\lambda) = 1\right] = 0$. This result along with Eq. (2), concludes the proof of Thm. 1. $\square$

Proof of Thm. 2. We give a formal proof that our predicate blind signature scheme $\mathsf{PBS}_{\mathsf{RSA}}$ from Fig. 4 satisfies unforgeability according to Def. 2 by providing reductions to the security properties of its building blocks. We proceed by a sequence of hybrid games.

GAME_0. This is game OMUF from Fig. 2 with PBS instantiated by $\mathsf{PBS}_{\mathsf{RSA}}$ from Fig. 4. The generic PBS.Stp hence is replaced by the setup from Fig. 4. The call of PBS.SSig in the oracle Sign is instantiated by doing the steps is the algorithm $\mathsf{PBS}_{\mathsf{RSA}}$.SSig in Fig. 4. We call this game also the real game.

GAME_1. In GAME_1 we modify the behavior of the oracle Sign. On each call with input (t, π), if the proof verification succeeds (i.e., PS.Vry outputs 1), we extract the values (m, r, κ) from π by invoking algorithm PS.Ext with inputs: the crs from par in sk, the trapdoor τ generated during PS.Stp (which was run as part of $\mathsf{PBS}_{\mathsf{RSA}}$.Stp), the statement $\mathbb{x} := (N, e, \mathsf{H}_{\mathsf{PSS}}, t)$ where $(N, e, \mathsf{H}_{\mathsf{PSS}})$ are from sk and t is from the query, and the proof π. We then verify that the extracted values satisfy two conditions: (1) $\mathsf{P}(\varphi, m) = 1$ and (2) $\mu\kappa^e \equiv t \pmod N$ where $\mu := \mathsf{H}_{\mathsf{PSS}}.\mathsf{Enc}(m, r)$. If either condition fails, we abort the game by returning $\perp$.

Reduction from PoK of PS. We now show that the difference between the advantage in the real game and in GAME_1 is bounded by the advantage in the proof of knowledge game POK (Def. 11) for $\mathsf{PS}[\mathsf{R}_{\mathsf{Blind}}]$. We construct an adversary $\mathcal{P}$ against POK as follows:

$\mathcal{P}$ receives as input the crs generated by PS.Stp. It then perfectly simulates the real game to $\mathcal{A}$, using the provided crs in par. Whenever Sign is queried with (t, π) and PS.Vry accepts π, $\mathcal{P}$ constructs the statement $\mathbb{x} := (N, e, \mathsf{H}_{\mathsf{PSS}}, t)$ and outputs the statement-proof pair $(\mathbb{x}, \pi)$ in the POK game. By definition of the POK game, $\mathcal{P}$ wins POK exactly when it outputs a statement-proof pair where verification passes but extraction fails to produce a valid witness. Observe that GAME_1 aborts precisely when there exists a statement-proof pair that verifies correctly but for which the extractor fails to recover a valid witness. This is exactly the winning condition for $\mathcal{P}$ in POK. Since $\mathcal{P}$ perfectly simulates the game for $\mathcal{A}$

until the point it receives the statement-proof pair, we have:

$$\mathsf{Adv}^{\mathrm{omuf}}_{\mathsf{PBS}_{\mathsf{RSA}}}(\mathcal{A},\lambda) \leq \mathsf{Adv}^{\mathrm{pok}}_{\mathsf{PS}[\mathsf{R}_{\mathsf{Blind}}]}(\mathcal{P},\lambda) + \Pr\left[\textsc{Game}_1^{\mathcal{A}}(\lambda)=1\right] \quad (3)$$

$\textsc{Game}_2$. In $\textsc{Game}_2$ we prepare the reduction to EUF-CMA security of the RSA-BSS signature scheme $\mathsf{RSA}[\mathsf{RSAGen},\mathsf{PSSGen}]$ underlying $\mathsf{PBS}_{\mathsf{RSA}}$, by making the following changes: First we introduce an empty list Σ.

Then we modify the behavior of the oracle Sign so that after extracting (m,r,κ), we compute a RSA signature on m by using the encoding randomness r under the signing key $\mathsf{sk}{:=}(N,e,\mathsf{H}_{\mathsf{PSS}})$ by running $\sigma{:=}\mathsf{PBS}_{\mathsf{RSA}}.\mathsf{SSig}(\mathsf{sk},m;r)$. Next, we prepare the output message $\mathsf{msg}_{\mathsf{out}}{:=}\sigma\kappa(\mathrm{mod} N)$ and before returning it to the adversary we store in Σ the pair (m,σ). The adversary now obtains the value $\mathsf{msg}_{\mathsf{out}}$ where σ by definition of RSA.Sig (Fig. 8) is equal to $\mathsf{H}_{\mathsf{PSS}}.\mathsf{Enc}(m,r)^d(\bmod N)$ and since the output of PS.Vry is 1 and PS.Ext successfully outputs (m,r,κ) (otherwise we abort, according to $\textsc{Game}_1$) we also have that $\mathsf{P}(\varphi,m)=1$ and $\mu\kappa^e \equiv t \pmod N$. That is, it immediately follows that the view of the adversary in $\textsc{Game}_2$ is distributed equivalently to its view in $\textsc{Game}_1$, thus we have that $\Pr\left[\textsc{Game}_1^{\mathcal{A}}(\lambda)=1\right]$ is equal to $\Pr\left[\textsc{Game}_2^{\mathcal{A}}(\lambda)=1\right]$.

REDUCTION OF EUF-CMA OF RSA TO $\textsc{Game}_2$. To finish the proof, we construct adversary $\mathcal{R}$ that succeeds in the game EUF-CMA against the RSA signature scheme $\mathsf{RSA}[\mathsf{RSAGen},\mathsf{PSSGen}]$ with the same probability $\Pr\left[\textsc{Game}_2^{\mathcal{A}}(\lambda)=1\right]$. By the definition of EUF-CMA, $\mathcal{R}$ receives as challenge input a RSA verification key $\mathsf{vk}{:=}(N,e,\mathsf{H}_{\mathsf{PSS}})$ and has access to a signing oracle Sign. The adversary first does the $\mathsf{PBS}_{\mathsf{RSA}}.\mathsf{Stp}$ by computing the common reference string crs for $\mathsf{PS}[\mathsf{R}_{\mathsf{Blind}}]$. Moreover, $\mathcal{R}$ initializes a list Σ used to store the message-signature pairs from its signing oracle Sign. When $\mathcal{R}$ simulates $\textsc{Game}_2$ for $\mathcal{A}$, it embeds its challenge RSA public key $(N,e,\mathsf{H}_{\mathsf{PSS}})$ into the verification key for $\mathsf{PBS}_{\mathsf{RSA}}$ (i.e., computes the verification key vk of $\mathsf{PBS}_{\mathsf{RSA}}$ by leveraging crs and the tuple $(N,e,\mathsf{H}_{\mathsf{PSS}})$). The corresponding secret key is not required since $\mathcal{R}$ on each Sign query by $\mathcal{A}$ forwards the call to its signing oracle Sign of EUF-CMA game. The simulation is perfect. We show that if $\mathcal{A}$ wins $\textsc{Game}_2$ outputting $\mathcal{F}=(m_i^*,\sigma_i^*)_{i\in[\ell]}$, then this set must contain a successful forgery for $\mathcal{R}$, that is, an element that is not contained in Σ.

Recall that in $\textsc{Game}_2$, for each signing query (t,π) where verification succeeds, we extract (m,r,κ) and check that $\mathsf{P}(\varphi,m)=1$ and $\mu\kappa^e \equiv t \pmod N$ where $\mu = \mathsf{H}_{\mathsf{PSS}}.\mathsf{Enc}(m,r)$. If these conditions hold, we obtain the signature of m from the RSA signing oracle and we return it (after blinding with κ) to $\mathcal{A}$. For $\mathcal{A}$ to win, it must output a set $\mathcal{F}=(m_i^*,\sigma_i^*)_{i\in[\ell]}$ of valid message-signature pairs where at least one message was never queried to the signing oracle. That is, for any winning pair $(m_j^*,\sigma_j^*)\in\mathcal{F}$ we have that $\mathsf{PBS}_{\mathsf{RSA}}.\mathsf{Vry}(\mathsf{vk},m_j^*,\sigma_j^*)=1$ and no explaining map f exists such that any predicate in $\mathbf{P}$ is valid for m_j^* . Consequently, for any such valid forgery (m_j^*,σ_j^*), we have that m_j^* was never queried to the signing oracle. This represents a valid forgery against the underlying RSA signature scheme. Therefore, $\mathcal{R}$ can simply output (m_j^*,σ_j^*) as its forgery in the EUF-CMA game against RSA.

Thus, we have that $\Pr\left[\textsc{Game}_1^{\mathcal{A}}(\lambda)=1\right]$ is at most equals to $\mathsf{Adv}^{\mathrm{euf\text{-}cma}}_{\mathsf{RSA}[\mathsf{RSAGen},\mathsf{PSSGen}]}(\mathcal{R},\lambda)$ and this along with prior considerations and Eq. (3) concludes the proof of Thm. 2. □